COMPAQ

WHITE PAPER

technical level

BASIC ADVANCED

October 2000

Prepared by
Alain Lissoir

Technology Consultant
Compaq Services

Compaq Computer
Corporation

The powerful combination of Windows Script Host and Active Directory Service Interfaces under Windows 2000

The purpose of this white paper is to explain some advanced scripting techniques using Windows Script Host (WSH) and Active Directory Service Interfaces (ADSI). Through the use of examples, the reader will learn how to write reusable code with WSH, how to execute queries, and how to manipulate Active Directory rights. The document also shows how to create miscellaneous object types such as different group types, user objects, and Exchange 2000 mailboxes.

This white paper is the second part of a series of which "Understanding Microsoft Windows Script Host and Active Directory Interfaces in Windows 2000" is the first part.

Acknowledgments:

Micky Balladelli
Jan De Clercq
Mike Dransfield
Andrew Gent
Andy Harjanto (Microsoft Corporation)
Dung Hoang Khac
Jurgen Klejdzinski
Tony Redmond
John Rhoton
Rudy Schockaert
Alan Smith (Microsoft Corporation)
Roland Schoenauen

NOTICE

The information in this publication is subject to change without notice.

Compaq Computer Corporation shall not be liable for technical or editorial errors or omissions contained herein, nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This publication does not constitute an endorsement of the product or products that were tested. The configuration or configurations tested or described may or may not be the only available solution. This test is not a determination of product quality or correctness, nor does it ensure compliance with any federal, state or local requirements. Compaq does not warrant products other than its own strictly as stated in Compaq product warranties.

October 2000

2nd version.

Feedback may be addressed directly to alain.lissoir@compaq.com

TABLE OF CONTENTS

TABLE OF SAMPLES

TABLE OF FIGURES

TABLE OF TABLES

INTRODUCTION

In Part 1 "Understanding Microsoft Windows Script Host and Active Directory Service Interfaces in Windows 2000", the reader has the opportunity to examine all the basic functions of WSH and ADSI. Basic access methods are explained via small samples. The second part of this study will demonstrate that the scripting technology can be more than a single script file development. The combination of WSH and ADSI gives a powerful tool to developers and administrators.

Before WSH, any specific Windows system customization required the use of a custom tool or an external solution such as Resource Kit utilities, Perl engines, and so on.

Today, under Windows 2000, using WSH and ADSI (and even possibly other COM objects written for automation languages), developers and administrators can handle most of their tasks from one unique environment: Windows Script Host.

One of the purposes of this White Paper is to develop a script capable of creating users (with their properties), their home directories, their profile directories (with file system right settings) and the Exchange 2000 mailbox (with access rights). The document will examine all steps in detail to reach that goal. All functions needed will be developed and commented on in this document.

This White Paper is intended for people with a good knowledge of Windows 2000, Exchange 2000, ADSI and WSH. It is strongly recommended that you read the first part of the study (and also other White Papers on Windows 2000 given in reference) if the reader is new to those technologies.

This white paper is based on Windows 2000 and Exchange 2000. Some changes may occur for later versions.

PREREQUISITES

The reader should have an understanding of the following technologies before reading this document:

- VB Scripting
- How to use COM objects inside VB Scripting
- Windows 2000 Active Directory
- Windows 2000 concepts
- Part 1 - Understanding Microsoft WSH and ADSI in Windows 2000

To run sample scripts given in this document, you must have:

- Windows 2000 installed with Active Directory installed
- Microsoft Excel 97 or Microsoft Excel 2000 installed
- Microsoft Exchange 2000 installed

WHAT'S NEW IN THIS UPDATED VERSION?

This update concerns the support for the commercial release of Exchange 2000.

Sample 18 - Creating groups for organizational unit members

The statement including the "EnableE-MailFunction.vbs" function has been changed to include the new function name "EnableE-MailFunction (ADSI).vbs".

Sample 20 - Enabling E-Mail on object for Exchange 2000 with ADSI

The Sample 20 didn't create correctly a mail-enabled recipient with ADSI. The script didn't initialize the ***targetAddress*** attribute of the mail-enabled object.

The function "EnableE-MailFunction.vbs" using ADSI to enable the mailbox has been renamed to:

"EnableE-MailFunction (ADSI).vbs"

This differentiates the ADSI function from the one using CDO for Exchange Management (CDOEXM) to enable the mailbox:

"EnableE-MailFunction (CDOEXM).vbs"

The function call creating the Exchange 2000 Security Descriptor has been moved inside the "EnableE-MailFunction (ADSI).vbs" function. The Exchange 2000 Security Descriptor creation is mandatory if we create the mailbox via ADSI. When creating the mailbox via CDOEXM, the CDOEXM COM object constructs a default security descriptor.

This function contained a wrong reference to the variable objUser. It should be objObject instead. Originally objUser and objObject referenced the same object, because objUser was a global variable the scripts worked correctly. But it was potential bug if the function caller was using another variable than objUser.

The "EnableE-MailFunction (ADSI).vbs" uses the new Exchange 2000 rights contants to construct the default Exchange 2000 default security descriptor (See Include.vbs)

Sample 21 - Enabling E-Mail on object for Exchange 2000 with CDOEXM

The sample is a new function to show how to use CDO for Exchange Management (CDOEXM) to mail-enable or mailbox-enable an object. This function is given for reference only.

Sample 22 - Creating users under Windows 2000 with their associated Exchange 2000 mailboxes

The statement including the "EnableE-MailFunction.vbs" function has been changed to include the new function name "EnableE-MailFunction (ADSI).vbs".

The Private Store name has been moved to a constant such as the Organization name, the Administration group name and the Storage group name. Before this parameter was hard coded in the script code and was not modifiable in the script header.

A function "CreateUserID" to create the Windows 2000 User Ids has been added to the script.

DEMONSTRATION BY SAMPLES

Rather than doing a long description of all the WSH and ADSI functionality, we will examine practical samples showing how to manipulate data from the Active Directory. Discussing different topics between each sample will show the different capabilities of the two technologies. Capabilities such as security configuration, code reusability, data search, Windows 2000 user creation, and Exchange 2000 mailboxes creation will be examined through these samples. Of course, the purpose of the document is not to provide all possible applications of WSH and ADSI. The purpose of the scripts is to give the reader a good idea of "how the things work".

Searching in the Active Directory

One approach to searching across the Active Directory could be by browsing the tree to find all existing objects and their associated properties. (See the first part of this study: "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000".) That is, of course, a valid method but it has some performance disadvantages. In addition, the code must contain the search condition in the script logic.

Another method is using a search mechanism provided by ADSI based on the **IDirectorySearch** interface. ADSI implements several interfaces but only one is suitable for use in a scripting environment such as Windows Script Host.

Here are more details about the available methods:

- **IDirectorySearch**

ADSI provides the **IDirectorySearch** interface to query Active Directory (as well as other directory services such as NDS) using LDAP. **IDirectorySearch** is a COM interface that returns richly typed data (Integer, Octet String, String, Security Descriptor, UTC-Time, Large Integer, Boolean, etc.). The **IDirectorySearch** interface provides a very high level and low overhead interface for querying data of a directory or a global catalog. The **IDirectorySearch** COM interface is not available to development environments that use Automation Languages (Visual Basic, VBScript, JavaScript and so on).

- **OLE DB**

OLE DB is a set of COM interfaces that provide applications with uniform access to data stored in diverse information sources, regardless of location or type. ADSI also provides an OLE DB provider enabling applications to use OLE DB to access Active Directory. The ADSI OLE DB provider uses the **IDirectorySearch** interfaces to submit queries to Active Directory and to collect the results. Because all are COM-based, this method of search is the most suitable from a scripting environment.

- **LDAP API**

Windows 2000 domain controllers are directory servers compliant with LDAP version 3. The LDAP API is a C-style function library. Applications can use the LDAP API to search Active Directory. Without an encapsulation of the API in a COM object, this type of service is not accessible from Windows Script Host.

Querying the Active Directory

To query the Active Directory, the Sample 1 uses the OLE DB techniques implemented by the set of COM interfaces provided by ADSI and ADO. The OLE DB entry point provided by ADSI can be used "transparently" in the scripts. (See the first part of this study: "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000" for an overview of the ADSI structure.)

Having a search function will simplify access to some objects or attributes in the Active Directory. By using a simple syntax, it is possible to obtain an **ADsPath** pointer to the searched object. More than a pointer, an attribute list can be retrieved by this search function.

Whatever the manner the search function is written in, a LDAP search operation needs four key elements:

1. The base object in which the search will start. (This is an object in the Active Directory such as an **organizationalUnit.** The script Sample 1 also accepts a naming context such as the **defaultNamingContext**, **SchemaNamingContext)**.

   If a specific naming context is passed, the function will resolve the context name given to the corresponding **ADsPath** (See Sample 1 on page 10).

2. The filter determines which elements have to be selected (based on conditions) from the Active Directory.

   The syntax for queries can be SQL syntax or LDAP syntax. Of course, SQL syntax is relevant for applications making SQL queries and Active Directory queries. In the next sample, the LDAP syntax query is used.

   Any accessible characteristics of an object can be used to make a query. Some examples are:

   - *(|(objectClass=domainDNS)(objectClass=organizationalUnit))*

   This will return the list of all the objectClass equal to **domainDNS** or equal to **organizationalUnit.** Note the '|' sign for the 'or' statement.

   - *(&(objectClass=user)(objectCategory=person))*

   This will return the list of all objectClass equal to **user** for which the objectCategory is equal to **person**. Actually this will provide a list of all users available in the selected naming context. In this example, the objectClass **contact** will be discarded, as it is also an objectCategory **person**. Note the '&' sign for the 'And' statement.

   To get both **contact** and **user** objectClass, following syntax can be use:

   - *(&(|(objectClass=user) (objectClass=contact)) (objectCategory=person))*

   or to make the things simpler (because an object class **user** or **contact** is an object category equal to **person**), use:

   - *(objectCategory=person)*

   In the first version note the possibility to combine search conditions '|' and '&'.

3. The attributes of the retrieved objects that are desired. This can be any attribute associated with the retrieved objects (**ADsPath**, **cn**, **distinguishedName**, …).

4. How deep the search has to run in the Active Directory.

Base: A **Base** search is limited to the base object selected. If the base object has children, they will not be included in the search. Only elements that are direct members of the base object are examined.

OneLevel: A **OneLevel** search is restricted to the immediate children of a base object, but excludes the base object itself. This scope is perfectly adapted for a search inside the **schemaNamingContext**. This is used in Sample 10 on page 29.

Subtree: A **Subtree** search includes the entire subtree below the base object.

Choosing the right scope is important for search performance.

Sample 1 below shows the complete search request formulation.

Sample 1: The Active Directory search function

```
' VB Script making search operations in AD through ADSI OLE DB interface via ADO '
'                                                                                 '
' Version 1.00 - Alain Lissoir                                                    '
' Compaq Computer Corporation - Professional Services - Belgium -                 '
'                                                                                 '
' Any comments or questions:                       EMail:alain.lissoir@compaq.com '

Option Explicit

' ------------------------------------------------------------------------------------------
Function ADSearch _
         (strNamingContextSearch, strFilterSearch, strAttribsToReturnSearch, _
                                                          strDepthSearch, boolEcho)

        Dim objDictionary

        Dim objRoot
        Dim strNamingContext
        Dim objNamingContext

        Dim objADOConnnection
        Dim objCommand
        Dim objRecordSet

        Dim strADsPathSearch

        Dim intIndice

        Set objDictionary = CreateObject ("Scripting.Dictionary")

        Set objADOConnnection = CreateObject("ADODB.Connection")
        objADOConnnection.Provider = "ADsDSOObject"
        objADOConnnection.Open "Active Directory Provider"

        Set objCommand = CreateObject("ADODB.Command")
        Set objCommand.ActiveConnection = objADOConnnection

        If Ucase (Mid (strNamingContextSearch, 1, 7)) = "LDAP://" Or _
           Ucase (Mid (strNamingContextSearch, 1, 5)) = "GC://" Then

           Set objNamingContext = GetObject(strNamingContextSearch)
        Else
           Set objRoot = GetObject("LDAP://RootDSE")
           strNamingContext = objRoot.Get(strNamingContextSearch)
           Set objRoot = Nothing

           ' Search is executed in LDAP: namespace.
           ' Search in Global Catalog (GC:) instead of classical directory (LDAP:) can
           ' increase search speed but be aware if attributes searched are not
           ' replicated in GC. (Such as data in the schema or some context attributes)
           Set objNamingContext = GetObject("LDAP://" & strNamingContext)
```

```
  52:         End If
  53:
  54:         strADsPathSearch = "<" & objNamingContext.ADsPath & ">"
  55:
  56:         objCommand.CommandText = strADsPathSearch & ";" & _
  57:                                  strFilterSearch & ";" & _
  58:                                  strAttribsToReturnSearch & ";" & _
  59:                                  strDepthSearch
  ..:
  65:         objDictionary.Add "RecordCount", objRecordSet.RecordCount
  66:
  67:         While Not objRecordSet.EOF
  68:             For intIndice = 0 To objRecordSet.Fields.Count - 1
  ..:
  76:                 objDictionary.Add objRecordSet.Fields(intIndice).Name & ":" & _
  77:                                   objRecordSet.AbsolutePosition & intIndice, _
  78:                                   objRecordSet.Fields(intIndice).Value
  79:             Next
  80:
  81:             If boolEcho Then WScript.Echo
  82:
  83:             objRecordSet.MoveNext
  84:         Wend
  ..:
  88:         Set ADSearch = objDictionary
  89:
  90:         Set objNamingContext = Nothing
  91:
  92:         WScript.DisconnectObject objCommand
  93:         Set objCommand = Nothing
  94:
  95:         WScript.DisconnectObject objADOConnnection
  96:         Set objADOConnnection = Nothing
  97:
  98:         WScript.DisconnectObject objDictionary
  99:         Set objDictionary = Nothing
 100:
 101:End Function
```

The access to the search engine provided by the ADSI OLE DB COM object is at line 31, 32, 33, 35 and 36. At line 33, note the name of the provider; that is where the reference to the ADSI OLE DB provider is made. On line 35, the object command is created to formulate the query. Line 36 links the command object to the current ADSI OLE DB connection.

On lines 38 to 52, the script binds to the selected context or **ADsPath** given. The function accepts two kinds of parameter: a **RootDSE** context name or an **ADsPath** beginning with LDAP// or GC://. In case of a context parameter, to have its **ADsPath**, the script binds to the **RootDSE** object to get the context **distinguishedName** attribute (line 43 to 51). The **ADsPath** determines the starting point of the search. The function accepts (as a parameter) one of the three available contexts from the **RootDSE.**

Note: To make the code easier to read, the test to validate the passed parameters is not included. The purpose is to show the basic principles of ADSI usage and not to provide production quality code.

The rest of the script is very simple. The only peculiarity is the need to build the right command syntax to formulate the query. The script builds a command syntax string. Note the **ADsPath** inclusion between '<>' (line 54). All other parts of the query command are concatenated together and separated by a semi-column ';' (line 56).

Here is an LDAP query syntax string as an example:

<LDAP://DC=w2k-home,DC=com>;(|(objectCategory=domainDNS)(objectClass=organizationalUnit));cn,name,distinguishedName;subTree

Next, the query is executed (line 63). Once the query completes, the results need to be returned. The 'Dictionary' object will help in this task.

The Dictionary Object returns the query results

The Dictionary Object is part of the Scripting Run-Time Library. (See Microsoft Visual Basic Scripting/Java Scripting on-line Documentation for more information about the Scripting Run-Time Library.) The dictionary is a "repository" where data of any kind can be stored. Moreover, the dictionary lets you retrieve the stored data based on a key. The key can be a number or a string. The key has to be **unique**. A script can fetch a complete list of all existing keys just by using a 'For Each' loop and some dictionary methods (see Sample 2 on page 13). The dictionary object is a like an array where a script can retrieve heterogeneous data associated to an ordinal position (or a string) used as key.

The ADSearch function builds a dictionary with all the data retrieved from the *RecordSet* containing the query result (line 76). This will makes it easier to return the query result to a routine calling the ADSearch function. In such case, only one parameter is returned: the dictionary object (line 88).

Building the dictionary object is easy. First the script must create an instance of the object (line 29). Once created, the dictionary is ready to be used. To add elements to the dictionary, the 'Add' method is used (line 76). Note that the first element added is the number of matches found for that particular query (key='RecordFound'). This will help the calling module to determine the number of 'RecordSet' found (line 65).

The 'Add' method has two parameters:

1. The key to be used in the dictionary.
2. The value to be inserted in the dictionary.

As it can be noticed in the ADSearch function, the script has a parameter called *boolEcho*. Setting this parameter to 'True' will echo any object found in the record set. This will ease the understanding and debugging of the search function. (The code is not included in Sample 1 to clarify the view)

Below is an example of calling the ADSearch function:

```
Set objResultList = ADSearch ("DefaultNamingContext", _
                              "(|(objectClass=domainDNS)(objectClass=organizationalUnit))", _
                              "name,distinguishedName", _
                              "subTree", _
                              True)
```

The following output will be produced (the last parameter, *boolEcho,* is set to True for internal echoing):

```
Searching ...

(10) name = w2k-home
(11) distinguishedName = DC=w2k-home,DC=com

(20) name = Domain Controllers
(21) distinguishedName = OU=Domain Controllers,DC=w2k-home,DC=com

(30) name = Company Users
(31) distinguishedName = OU=Company Users,DC=w2k-home,DC=com

(40) name = New Marketing Division
(41) distinguishedName = OU=New Marketing Division,OU=Company Users,DC=w2k-home,DC=com

(50) name = Marketing
(51) distinguishedName = OU=Marketing,OU=Company Users,DC=w2k-home,DC=com

(60) name = Technical
(61) distinguishedName = OU=Technical,OU=Company Users,DC=w2k-home,DC=com

(70) name = Financial
(71) distinguishedName = OU=Financial,OU=Company Users,DC=w2k-home,DC=com
```

```
(80) name = Management
(81) distinguishedName = OU=Management,OU=Company Users,DC=w2k-home,DC=com

8 record(s) found.
```

The keys (with the associated data) used for the dictionary have the following form:

Dictionary Keys	Dictionary Data
name:10	w2k-home
distinguishedName:11	DC=w2k-home,DC=com
name:20	Domain Controllers
distinguishedName:21	OU=Domain Controllers,DC=w2k-home,DC=com
name:30	Company Users
distinguishedName:31	OU=Company Users,DC=w2k-home,DC=com
name:40	New Marketing Division
distinguishedName:41	OU=New Marketing Division,OU=Company Users,DC=w2k-home,DC=com
name:50	Marketing
distinguishedName:51	OU=Marketing,OU=Company Users,DC=w2k-home,DC=com
name:60	Technical
distinguishedName:61	OU=Technical,OU=Company Users,DC=w2k-home,DC=com
name:70	Financial
distinguishedName:71	OU=Financial,OU=Company Users,DC=w2k-home,DC=com
name:80	Management
distinguishedName:81	OU=Management,OU=Company Users,DC=w2k-home,DC=com

The name of the retrieved attribute is associated with the dictionary key. This simplifies the calling module to retrieve the attribute name. Because the dictionary key has to be unique, the script associates the *RecordSet.AbsolutePosition* and *RecordSet.Fields* in the key composition (line 76). This guarantees the uniqueness of the key. Between the attribute name and the *RecordSet.AbsolutePosition*, a column ':' is inserted to make it easier for the caller to split the string. (See line 18 of Sample 2)

The complete data set returned by the ADSearch function is in the dictionary object. It is the calling module's responsibility to look inside the dictionary to extract the query results. The following sample code calls the ADSearch function and then extracts the query results from the dictionary object.

Sample 2: The main function calling the ADSearch function.

```
 1:' VB Script to show how to use the ADSearch function within the same script     '
 2:'                                                                               '
 3:' Version 1.00 - Alain Lissoir                                                  '
 4:' Compaq Computer Corporation - Professional Services - Belgium -               '
 5:'                                                                               '
 6:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 7:
 8:Option Explicit
 9:
10:Dim objResultList
11:Dim objResult
12:Dim intColumnPosition
13:
14:Set objResultList = ADSearch ("DefaultNamingContext", _
15:                              "(|(objectClass=domainDNS)(objectClass=organizationalUnit))", _
16:                              "name,distinguishedName", _
17:                              "subTree", _
18:                              False)
19:WScript.Echo
20:WScript.Echo "Number of record found is " & objResultList.Item ("RecordCount")
21:WScript.Echo "Number of elements in the Scripting Dictionary object is " & objResultList.Count
22:
23:For Each objResult in objResultList
```

```
  24:    intColumnPosition = InStr (objResult, ":")
  25:    If intColumnPosition Then
  26:       Wscript.Echo Mid (objResult, 1, intColumnPosition - 1) & "=>" & _
  27:                        objResultList.Item (objResult)
  28:    End If
  29:Next
  ..:
  ..:
  ..:
```

Running that code gives the following output from the caller routine (the internal Echo is set to False) :

```
Number of record found is 8
Number of elements in the Scripting Dictionary object is 17
name=>w2k-home
distinguishedName=>DC=w2k-home,DC=com
name=>Domain Controllers
distinguishedName=>OU=Domain Controllers,DC=w2k-home,DC=com
name=>Company Users
distinguishedName=>OU=Company Users,DC=w2k-home,DC=com
name=>New Marketing Division
distinguishedName=>OU=New Marketing Division,OU=Company Users,DC=w2k-home,DC=com
name=>Marketing
distinguishedName=>OU=Marketing,OU=Company Users,DC=w2k-home,DC=com
name=>Technical
distinguishedName=>OU=Technical,OU=Company Users,DC=w2k-home,DC=com
name=>Financial
distinguishedName=>OU=Financial,OU=Company Users,DC=w2k-home,DC=com
name=>Management
distinguishedName=>OU=Management,OU=Company Users,DC=w2k-home,DC=com
```

This versatile Active Directory search function needs now to be made reusable for other scripts and languages.

Code reusability

The code for a function in an external file is not reusable under WSH 1.0. The only way to reuse the function is to copy and paste the code to include it in the new .VBS file. Unfortunately, this will generate several copies of the same function. Under Windows 2000, and with WSH 2.0, some new possibilities are offered to the programmer.

By file inclusion

WSH 2.0 permits miscellaneous script inclusions even if different languages are mixed. This possibility is realized by encapsulating the core main function (the caller) in an XML definition. The XML definition lets you define a file inclusion and the language used in the included script. This XML file is called a Windows Script File. (.WSF extension)

The next sample contains the same ADSearch caller code as in Sample 2 on page 13, except that two different versions of that code are presented. The first version is as before in VB Script (line 9 to 41). The second version is in Java script (line 43 to 77). They are both in different <job></job> identifiers.

Sample 3: Making an external VB Script function reusable from other scripting languages such as Java with XML encapsulation.

```
  1:' VB Script to show how to use the ADSearch from VB Script and Java script '
  2:'                                                                          '
  3:' Version 1.00 - Alain Lissoir                                             '
  4:' Compaq Computer Corporation - Professional Services - Belgium -          '
  5:'                                                                          '
  6:' Any comments or questions:               EMail:alain.lissoir@compaq.com '
  7:
  8:<Package>
  9:<job id="ADSearchVBSCaller">
 10:
 11:        <script language="VBScript" src="ADSearchFunction.vbs" />
 12:
```

```
13:         <script language="VBScript">
14:
15:         Option Explicit
..:
21:         Set objResultList = ADSearch ("DefaultNamingContext", _
22:                                       "(|(objectClass=domainDNS)(objectClass=organizationalUnit))",_
23:                                       "name,distinguishedName", _
24:                                       "subTree", _
25:                                       False)
26:
27:         WScript.Echo
28:         WScript.Echo "Number of record found is " & objResultList.Item ("RecordCount")
29:         WScript.Echo "Number of elements in the Scripting Dictionary object is " & _
30:                      objResultList.Count
31:
32:         For Each objResult in objResultList
33:             intColumnPosition = InStr (objResult, ":")
34:             If intColumnPosition Then
35:                Wscript.Echo Mid (objResult, 1, intColumnPosition - 1) & "=>" & _
36:                                                          objResultList.Item (objResult)
37:             End If
38:         Next
39:
40:         </script>
41:</job>
42:
43:<job id="ADSearchJSCaller">
44:
45:         <script language="VBScript" src="ADSearchFunction.vbs" />
46:
47:         <script language="JScript">
..:
54:         objResultList = ADSearch ("DefaultNamingContext",
55:                                   "(|(objectClass=domainDNS)(objectClass=organizationalUnit))",
56:                                   "name,distinguishedName",
57:                                   "subTree",
58:                                   false);
59:
60:         WScript.Echo ("Number of record found is " + objResultList.Item ("RecordCount"));
61:         WScript.Echo ("Number of elements in the Scripting Dictionary object is " & _
62:                        objResultList.Count);
63:
64:         enumResultList = new Enumerator (objResultList)
65:
66:         for (;!enumResultList.atEnd();enumResultList.moveNext())
67:             {
68:             intColumnPosition =  enumResultList.item().indexOf (":");
69:             if (intColumnPosition > 0)
70:                {
71:                WScript.Echo (enumResultList.item().substr (0, intColumnPosition) + "=>"
72:                                       + objResultList.Item (enumResultList.item()));
73:                }
74:             }
75:
76:         </script>
77:</job>
78:</Package>
```

Both versions will produce the same output result. More than making a simple reuse of the ADSearch function, the Windows Script File is also shared between different languages. Of course, including two functions in different languages doing the same process in a single Windows Script file does not make much sense. But including different scripts inside the same Windows Script file and the ability to call miscellaneous functions written in different languages is really interesting. Inside one Windows Script file, it is possible to group all the scripts needed for the administration or logon process.

To start the script job, the following DOS prompt command must be used:

*Cscript //Job:ADSearch**VBS**Caller TestADSearchFunction.wsf* to run the VB Script version.

*Cscript //Job:ADSearch**JS**Caller TestADSearchFunction.wsf* to run the Java Script version.

In Sample 3 above, each script version is included inside a <Job></job> marker. The <Job></job> has its associated identifier. For each job, the inclusion of the ADSearch Function is made by referencing the language used and the file containing the function (line 11 and 45).

A simple change in the script by inserting some XML markers produces a powerful result.

By COM object registration

Is it possible to make the ADSearch function a COM object? Yes, it is possible, but for this some changes to the ADSearch script code have to be made. This special type of script is called a Windows Script Component (.WSC extension).

To publish the script as a COM object, a set of information must be provided:

1. The COM object is a binary file, so a well-known binary file must be doing the job for the script.
2. To build a COM object, available interfaces and methods must be exposed.
3. To register the COM object, a registration and CLSID are needed.

In the sample, some references to existing COM objects are used (i.e. ADSI OLE DB Provider). Windows Script Components (WSC) work in the same way. Each time a .WSC script is registered, a pointer in the registry to a COM compliant binary object called SCROBJ.DLL is created. This creates the definition of the WSC in the system.

The .WSC file can be easily registered from the Windows NT Explorer as shown on Figure 1 below.

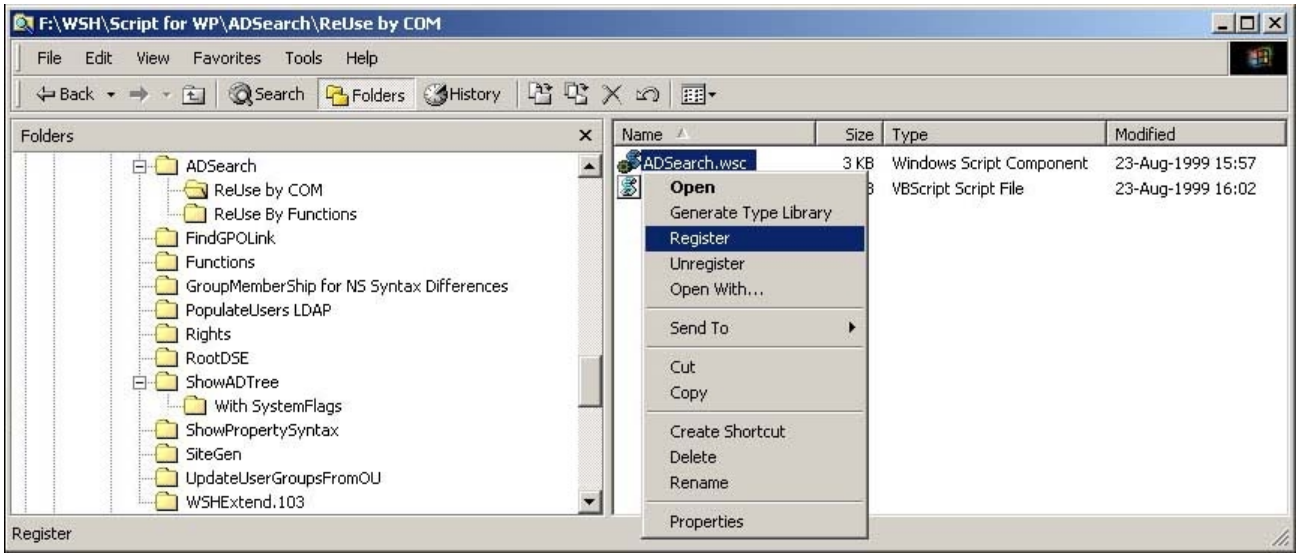


Figure 1 Registration of a Windows Script Component

Once the registration is complete, a registry entry is created for the ADSearch.WSC.

```
1:Windows Registry Editor Version 5.00
2:
3:[HKEY_CLASSES_ROOT\CLSID\{4ABB10F7-5307-11D3-8E5B-0008C7A92756}]
4:@="ADSearch"
5:
6:[HKEY_CLASSES_ROOT\CLSID\{4ABB10F7-5307-11D3-8E5B-0008C7A92756}\InprocServer32]
7:@="C:\\WINNT\\System32\\scrobj.dll"
8:"ThreadingModel"="Apartment"
9:
10:[HKEY_CLASSES_ROOT\CLSID\{4ABB10F7-5307-11D3-8E5B-0008C7A92756}\ProgID]
11:@="ADSearch.WSC.1.00"
12:
13:[HKEY_CLASSES_ROOT\CLSID\{4ABB10F7-5307-11D3-8E5B-0008C7A92756}\ScriptletURL]
14:@="file://C:\\Data\\WSH\\Script for WP\\ADSearch\\ReUse by COM\\ADSearch.wsc"
15:
16:[HKEY_CLASSES_ROOT\CLSID\{4ABB10F7-5307-11D3-8E5B-0008C7A92756}\VersionIndependentProgID]
17:@="ADSearch.WSC"
```

Of course, a CLSID must be provided to make the registration. To assist in this task, Microsoft provides a Wizard called the Windows Script Component Wizard.

This Wizard helps to build a .WSC core file with all the requested data for a COM object definition:

- CLSID
- Properties definition with variables mapping
- Methods definition

A Windows Script component is created through four important steps in the Windows Script Component Wizard. During step 1, the WSC Name and its associated ProgID are given. During step 2, the language used by the script must be specified. Step 3 and 4 are related to the definition of the properties and methods available from the script. The names entered during step 3 and 4 will be the reference to properties and methods used from any automation client using the COM object.

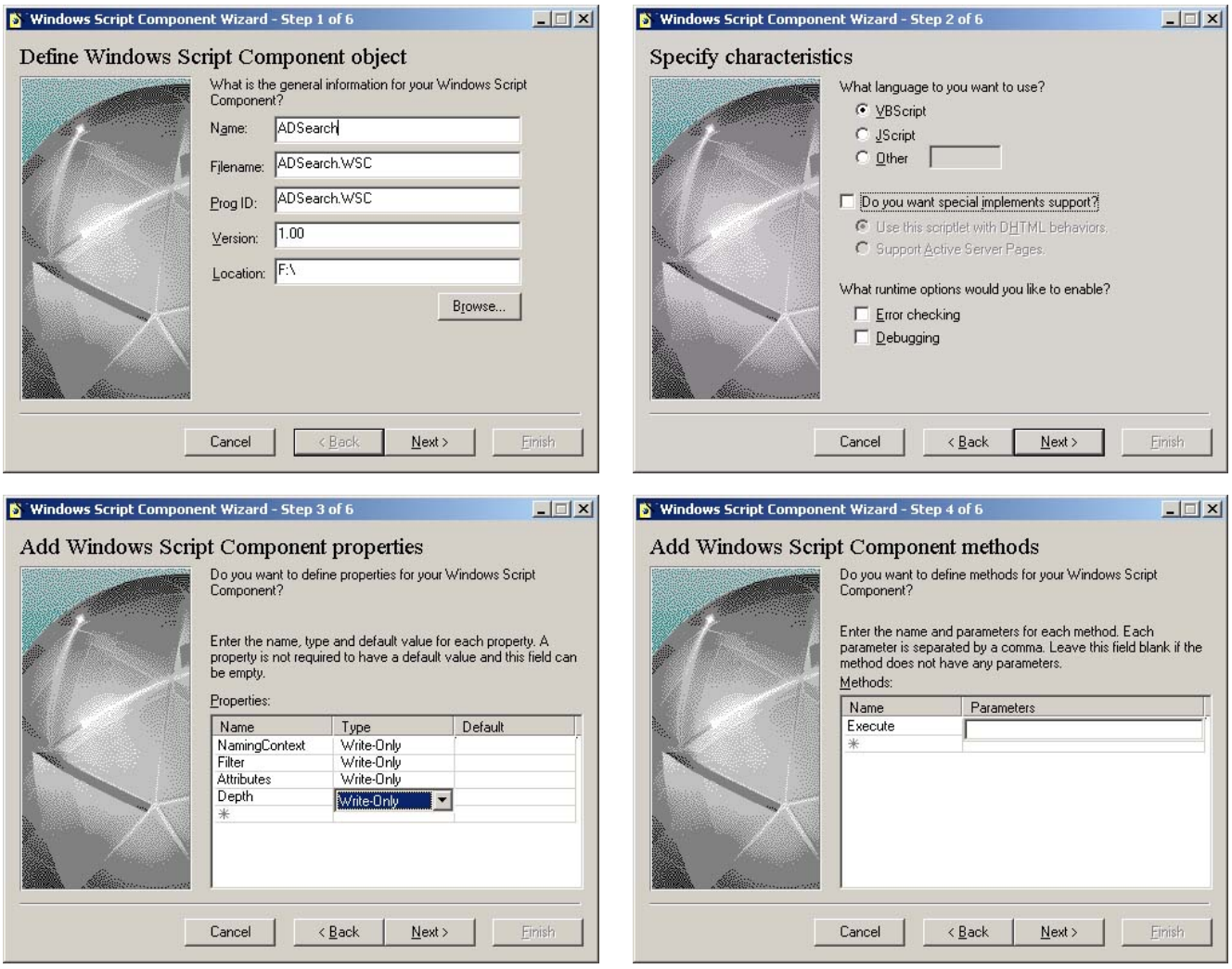


Figure 2 The Windows Script Component Wizard.

The "dummy" ADSearch.WSC that the wizard generates is ready to receive the ADSearch function. Of course, the generated file needs to be adapted in order to suit the input parameters used for the ADSearch function. Before examining those changes, Sample 4 shows the "dummy" ADSearch Windows Script Components generated by the Wizard. Note the CLSID in the registration section generated by the Wizard. (Line 8)

Sample 4: The XML dummy file generated by the Wizard Script Component.

```
1:<?xml version="1.0"?>
2:<component>
3:
4:<registration
5:      description="ADSearch"
6:      progid="ADSearch.WSC"
7:      version="1.00"
8:      classid="{4abb10f7-5307-11d3-8e5b-0008c7a92756}"
9:>
```

```
10:</registration>
11:
12:<public>
13:     <property name="NamingContext">
14:             <put/>
15:     </property>
16:     <property name="Filter">
17:             <put/>
18:     </property>
19:     <property name="Attributes">
20:             <put/>
21:     </property>
22:     <property name="Depth">
23:             <put/>
24:     </property>
25:     <method name="Execute">
26:     </method>
27:</public>
28:
29:
30:
31:<script language="VBScript">
32:<![CDATA[
33:
34:Dim NamingContext
35:Dim Filter
36:Dim Attributes
37:Dim Depth
38:
39:Function put_NamingContext(newValue)
40:     NamingContext = newValue
41:End Function
42:
43:Function put_Filter(newValue)
44:     Filter = newValue
45:End Function
46:
47:Function put_Attributes(newValue)
48:     Attributes = newValue
49:End Function
50:
51:Function put_Depth(newValue)
52:     Depth = newValue
53:End Function
54:
55:Function Execute()
56:     Execute = "Temporary Value"
57:End Function
58:
59:]]>
60:</script>
61:
62:</component>
```

The XML file that the wizard creates has the following sections:

1. **<Components></Components> section:**
   This section (lines 2 to 62) is equivalent to the <job></job> section seen in Sample 3. This element encloses the entire source code. As for the <job></job> element; an identifier attribute can be specified. This id attribute is only mandatory when several components are included in the same file.

2. **<Registration></Registration> section:**
   This section (lines 4 to 10) contains all the information necessary to perform a correct registration of the script as a COM object. All data entered during step 2 of the Wizard is entered in this section. The only new element in this section is the CLSID generated by the Wizard (line 8).

3. **<Public></Public> section**
   This section (lines 12 to 27) defines the properties and methods available from the module. In step 3, all the properties were defined as Write-Only. This is why there is only the <put/> statement. For Read-Only properties, the <get/> statement is used instead. Both can be combined for read and write properties. The <Public></Public> section always contains the <property></property> and <method></method> sections. Sample 5 below shows how to enhance and correlate the selected names with the variable originally used by the ADSearch function (Sample 1 on page 10).

4. **<Script></Script> section**
   This section (lines 31 to 60) has the same role as in the Windows Script File in Sample 3 on page 14. It encapsulates the script code to be executed and defines the script language in order to invoke the right script engine.

5. **<![CDATA[…]]> section**
   This is the character data delimiter. (Lines 32 and 59) This delimiter guarantees the XML 1.0 compliance (line 1). VB Script uses some characters (such as '&') that will confuse the XML parser because the XML parser expects version 1.0 compliant coding. The CDATA delimiter tells the XML parser to skip parsing operations on that piece of code.

   The ADSearch function will be inserted into the CDATA section. From lines 34 to 57, the "dummy" implementation of each property setting and method is included. The final ADSearch Windows Script Component shows how to interface the existing ADSearch function with these COM interfaces and methods.

Sample 5: The ADSearch VB Script function exposed as a COM component with Windows Script Component.

```
 1:<?xml version="1.0"?>
 2:
 3:<!-- Windows Script Component exposing the ADSearch function as a COM reusable -->
 4:<!-- object.                                                                    -->
 5:<!--                                                                           -->
 6:<!-- Version 1.00 - Alain Lissoir                                              -->
 7:<!-- Compaq Computer Corporation - Professional Services - Belgium -           -->
 8:<!--                                                                           -->
 9:<!-- Any comments or questions:                EMail:alain.lissoir@compaq.com -->
10:
11:<component>
12:
13:<registration
14:        description="ADSearch"
15:        progid="ADSearch.WSC"
16:        version="1.00"
17:        classid="{4abb10f7-5307-11d3-8e5b-0008c7a92756}"
18:>
19:</registration>
20:
21:<public>
22:        <property name="NamingContext" internalname="strNamingContextSearch">
23:                <put/>
24:        </property>
25:        <property name="Filter" internalname="strFilterSearch">
26:                <put/>
27:        </property>
28:        <property name="Attributes" internalname="strAttribsToReturnSearch">
29:                <put/>
30:        </property>
31:        <property name="Depth" internalname="strDepthSearch">
32:                <put/>
33:        </property>
34:        <method name="Execute">
35:        </method>
36:</public>
..:
40:<script language="VBScript">
41:<![CDATA[
42:
```

```
 43:Option Explicit
 44:
 45:Dim strNamingContextSearch
 46:Dim strFilterSearch
 47:Dim strAttribsToReturnSearch
 48:Dim strDepthSearch
 49:
 50:Function Put_strNamingContextSearch (newValue)
 51:         strNamingContextSearch = newValue
 52:End Function
 53:
 54:Function Put_strFilterSearch (newValue)
 55:         strFilterSearch = newValue
 56:End Function
 57:
 58:Function Put_strAttribsToReturnSearch (newValue)
 59:         strAttribsToReturnSearch = newValue
 60:End Function
 61:
 62:Function Put_strDepthSearch (newValue)
 63:         strDepthSearch = newValue
 64:End Function
 65:
 66:Function Execute()
 ..:
 82:        Set objDictionary = CreateObject ("Scripting.Dictionary")
 83:
 84:        Set objADOConnnection = CreateObject("ADODB.Connection")
 85:        objADOConnnection.Provider = "ADsDSOObject"
 86:        objADOConnnection.Open "Active Directory Provider"
 87:
 88:        Set objCommand = CreateObject("ADODB.Command")
 89:        Set objCommand.ActiveConnection = objADOConnnection
 90:
 91:        If Ucase (Mid (strNamingContextSearch, 1, 7)) = "LDAP://" Or _
 92:           Ucase (Mid (strNamingContextSearch, 1, 5)) = "GC://" Then
 93:
 94:           Set objNamingContext = GetObject(strNamingContextSearch)
 95:        Else
 96:           Set objRoot = GetObject("LDAP://RootDSE")
 97:           strNamingContext = objRoot.Get(strNamingContextSearch)
 98:           Set objRoot = Nothing
 ..:
104:           Set objNamingContext = GetObject("LDAP://" & strNamingContext)
105:        End If
...:
108:        strADsPathSearch = "<" & objNamingContext.ADsPath & ">"
109:
110:        objCommand.CommandText = strADsPathSearch & ";" & _
111:                                 strFilterSearch & ";" & _
112:                                 strAttribsToReturnSearch & ";" & _
113:                                 strDepthSearch
114:
115:        Set objRecordSet = objCommand.Execute
116:
117:        objDictionary.Add "RecordCount", objRecordSet.RecordCount
118:
119:        While Not objRecordSet.EOF
120:            For intIndice = 0 To objRecordSet.Fields.Count - 1
121:                objDictionary.Add objRecordSet.Fields(intIndice).Name & ":" & _
122:                                  objRecordSet.AbsolutePosition & intIndice, _
123:                                  objRecordSet.Fields(intIndice).Value
124:            Next
125:
126:            objRecordSet.MoveNext
127:        Wend
128:
129:        Set Execute = objDictionary
```

```
...:
136:End Function
137:
138:]]>
139:</script>
140:
141:</component>
```

To use the ADSearch function as Windows Script Component, some changes must be made to the "dummy" .WSC file. Here are the changes:

- Line 22, 25, 28 and 31

The statement "InternalName" is added to map the exposed property to the variable used internally in ADSearch. This will permit the use of the original variable name and expose a friendlier name for the COM properties.

property name="NamingContext" is mapped to the internal name variable "strNamingContextSearch"
property name="Filter" is mapped to the internal name variable "strFilterSearch"
property name="Attributes" is mapped to the internal name variable "strAttribsToReturnSearch"
property name="Depth" is mapped to the internal name variable "strDepthSearch"

- Line 45 to 48 and line 51, 55, 59 and 63

As with any other traditional function, it is necessary to declare the internal variables (line 45 to 48). When the exposed property name is assigned, the internal variable name must be updated. This is the purpose of the property function. The passed value is assigned to internal variable name (lines 51, 55, 59 and 63).

- Line 66

This is the location where the ADSearch function is inserted. No particular change to the original code is made except one: In the original code, the 'Wscript.Echo' statement was used for debugging purposes (See Sample 1 on page 10). It is important to know that the VB Script will run in a different context than WSH. The code will run in the SCROBJ.DLL context. In this context WSH base objects are not available. Two choices are possible:

1. Create an instance of the Wscript object.
2. Remove any reference to the Wscript object and use another method for debugging purposes (Event Sinking).

The second option was chosen and as consequence the 'boolEcho' parameter from the original function is removed. The 'boolEcho' functionality will be implemented by event sinking (See Sample 8 on page 10).

Now, the ADSearch function can be called from any type of client application. The key condition is to use a client application written in a language supporting automation such as VB, Java or Perl. The function is reusable, but this reusability suggests some enhancements for a production usage. Here are only two suggested enhancements:

- One enhancement to the ADSearch function is to provide parameter checking to protect the function against any invalid property (parameters) entered. This will avoid any curious or wrong behavior if the parameters given are not correct.
- An error handling function can be added to manage any possible error.

It was possible to add these enhancements from the beginning, but to keep the code as simple and readable as possible, it was decided to keep this out of the examples.

To complete the topic about the ADSearch COM object, the next two code samples show how to use the object: the first sample is written in VB Script and the other in Java script.

Note the small change made in comparison to Sample 2 on page 13 above. Line 16 is creating the ADSearch object instance. Lines 18 to 21 set the properties (pass the parameters). On line 23, the invocation of the method is executed.

Sample 6: Calling the ADSearch Windows Script Component from VB Script.

```
 1:' VB Script to show how to use the ADSearch function from a Windows Script      '
 2:' component.                                                                     '
 3:'                                                                                '
 4:' Version 1.00 - Alain Lissoir                                                   '
 5:' Compaq Computer Corporation - Professional Services - Belgium -                '
 6:'                                                                                '
 7:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 8:
 9:Option Explicit
10:
11:Dim objADSearch
12:Dim objResultList
13:Dim objResult
14:Dim intColumnPosition
15:
16:Set objADSearch = CreateObject("ADSearch.WSC")
17:
18:objADSearch.NamingContext = "DefaultNamingContext"
19:objADSearch.Filter = "(|(objectClass=domainDNS)(objectClass=organizationalUnit))"
20:objADSearch.Attributes = "name,distinguishedName"
21:objADSearch.Depth = "subTree"
22:
23:Set objResultList = objADSearch.Execute
24:
25:WScript.Echo
26:WScript.Echo "Number of record found is " & objResultList.Item ("RecordCount")
27:WScript.Echo "Number of elements in the Scripting Dictionary object is " & objResultList.Count
28:
29:For Each objResult in objResultList
30:    intColumnPosition = InStr (objResult, ":")
31:    If intColumnPosition Then
32:       Wscript.Echo Mid (objResult, 1, intColumnPosition - 1) & _
33:                    "=>" & objResultList.Item (objResult)
34:    End If
35:Next
```

Sample 7 is exactly the same script as Sample 6 but written in Java script. No particular remark needs to be made concerning this sample. It is the same logic as the VB script sample.

Sample 7: Calling the ADSearch Windows Script Component from Java Script.

```
 1:/*
 2:
 3: Java Script to show how to use the ADSearch function from a Windows Script
 4: component.
 5:
 6: Version 1.00 - Alain Lissoir
 7: Compaq Computer Corporation - Professional Services - Belgium -
 8:
 9: Any comments or questions:                EMail:alain.lissoir@compaq.com
10:
11:*/
12:
13:var objADSearch;
14:var objResultList;
15:var objResult;
16:var intColumnPosition;
17:var enumResultList
18:
19:objADSearch = new ActiveXObject("ADSearch.WSC");
20:
21:objADSearch.NamingContext = "DefaultNamingContext";
22:objADSearch.Filter = "(|(objectClass=domainDNS)(objectClass=organizationalUnit))";
23:objADSearch.Attributes = "name,distinguishedName";
24:objADSearch.Depth = "subTree";
25:
```

```
26:objResultList = objADSearch.Execute();
27:
28:WScript.Echo ("Number of record found is " + objResultList.Item ("RecordCount"));
29:WScript.Echo ("Number of elements in the Scripting Dictionary object is " + objResultList.Count);
30:
31:enumResultList = new Enumerator (objResultList)
32:
33:for (;!enumResultList.atEnd();enumResultList.moveNext())
34:    {
35:    intColumnPosition =  enumResultList.item().indexOf (":");
36:    if (intColumnPosition > 0)
37:       {
38:       WScript.Echo (enumResultList.item().substr (0, intColumnPosition) + "=>"
39:                                                  + objResultList.Item (enumResultList.item()));
40:       }
41:    }
```

Events sinking

Event sinking is a technology that implements mechanisms to create events inside the Windows Script Components. The calling module can capture (sink) these events via a specific function reference during the object creation. This specific function will capture the events from Windows Script Component object.

Sample 5 on page 20 can be used as it is. Only a few lines need to be added to enable event support in the Windows Script Component and in the Windows Script File invoking the Execute method.

This can be done in four steps:

1. In the <public></public> section of the Windows Script File, the following lines are added:

   ```
   <event name="Debug">
         <parameter name="Value" />
   </event>
   ```

   This new element in the section defines the event handler function name and the variable name to be used as the parameter to the event handler function. The event handler function will be added in the script invoking the Windows Script Component; in this case, the Windows Script File.

2. In the Execute Function code of the Windows Script Component, the following lines are added:

   ```
   FireEvent "Debug", objRecordSet.Fields(intIndice).Name & ":" & _
   objRecordSet.AbsolutePosition & intIndice & "=" & _
   objRecordSet.Fields(intIndice).Value
   ```

   To fire the event, the “FireEvent” function is used. Two parameters are needed: the event name defined in the <public></public> section and the value assigned to the parameter named ‘Value’ at step 1. In the example, the value assigned to the Dictionary object will be passed to the event.

3. In the Windows Script File (the caller), the following line is added:

   ```
   Set objADSearch = Wscript.CreateObject("ADSearch.WSC", "ADSearch_")
   ```

   In the Windows Script File, a reference between the event handler and the event name (defined in the Windows Script Component) must be realized. This task is completed during the ADSearch object creation by adding a new parameter; which is the “main” label used for the event handler function. In this case: “ADSearch_”. Because, the main script needs to support event handling from the WSC script, you must

specify the *Wscript.CreateObject* method rather than the *CreateObject* native method of the VB script engine. This is because the *Wscript.CreateObject* method lets you specify the additional parameter.

4. In the Windows Script File (the caller), an event handler function is added:

```
' Event Handler
sub ADSearch_Debug(Value)
   WScript.Echo "Debug: " & Value
end sub
```

This last change is the addition of the event handler function. This is a standard function definition. The only restriction is that the function must match the prior definitions made for the event. The function must be named by combining the names given at step 1 and step 3 (Event name and Main function name). The parameter of the function is referenced as given during step 1.

In Sample 2 (on page 13) and in Sample 3 (on page 14), the last parameter of the function (*boolEcho*) enabled a debug mode for the ADSearch function by echoing all the assigned values to the Dictionary object. By implementing an event in this ADSearch object, it is possible to obtain the same result. If the event handler is defined in the calling module, all assigned values to the Dictionary object will be echoed as shown in the output sample below.

```
Debug: name:10=w2k-home
Debug: distinguishedName:11=DC=w2k-home,DC=com
Debug: name:20=Home Business
Debug: distinguishedName:21=OU=Home Business,DC=w2k-home,DC=com
Debug: name:30=Accounting
Debug: distinguishedName:31=OU=Accounting,OU=Home Business,DC=w2k-home,DC=com
Debug: name:40=Technical
Debug: distinguishedName:41=OU=Technical,OU=Home Business,DC=w2k-home,DC=com
Debug: name:50=Management
Debug: distinguishedName:51=OU=Management,OU=Home Business,DC=w2k-home,DC=com
Debug: name:60=Marketing
Debug: distinguishedName:61=OU=Marketing,OU=Home Business,DC=w2k-home,DC=com
Debug: name:70=Assistants
Debug: distinguishedName:71=OU=Assistants,OU=Marketing,OU=Home Business,DC=w2k-home,DC=com
Debug: name:80=Domain Controllers
Debug: distinguishedName:81=OU=Domain Controllers,DC=w2k-home,DC=com

Number of record found is 8
Number of elements in the Scripting Dictionary object is 17
name=>w2k-home
distinguishedName=>DC=w2k-home,DC=com
name=>Home Business
distinguishedName=>OU=Home Business,DC=w2k-home,DC=com
name=>Accounting
distinguishedName=>OU=Accounting,OU=Home Business,DC=w2k-home,DC=com
name=>Technical
distinguishedName=>OU=Technical,OU=Home Business,DC=w2k-home,DC=com
name=>Management
distinguishedName=>OU=Management,OU=Home Business,DC=w2k-home,DC=com
name=>Marketing
distinguishedName=>OU=Marketing,OU=Home Business,DC=w2k-home,DC=com
name=>Assistants
distinguishedName=>OU=Assistants,OU=Marketing,OU=Home Business,DC=w2k-home,DC=com
name=>Domain Controllers
distinguishedName=>OU=Domain Controllers,DC=w2k-home,DC=com
```

To complete this topic, here are the Windows Script Component (below) and the Windows Script File (on page 28) code samples.

Sample 8: Windows Script Component triggering event

```
 1:<?xml version="1.0"?>
 2:
 3:<!-- Windows Script Component exposing the ADSearch function as a COM reusable -->
 4:<!-- object. (With event sinking)                                               -->
 5:<!--                                                                            -->
 6:<!-- Version 1.00 - Alain Lissoir                                               -->
 7:<!-- Compaq Computer Corporation - Professional Services - Belgium -            -->
 8:<!--                                                                            -->
 9:<!-- Any comments or questions:                EMail:alain.lissoir@compaq.com -->
10:
11:<component>
12:
13:<registration description="ADSearch"
14:              progid="ADSearch.WSC"
15:              version="1.00"
16:              classid="{4abb10f7-5307-11d3-8e5b-0008c7a92756}">
17:</registration>
18:
19:<public>
20:        <property name="NamingContext" internalname="strNamingContextSearch">
21:                <put/>
22:        </property>
23:
24:        <property name="Filter" internalname="strFilterSearch">
25:                <put/>
26:        </property>
27:
28:        <property name="Attributes" internalname="strAttribsToReturnSearch">
29:                <put/>
30:        </property>
31:
32:        <property name="Depth" internalname="strDepthSearch">
33:                <put/>
34:        </property>
35:
36:        <method name="Execute">
37:        </method>
38:
39:        <event name="Debug">
40:                <parameter name="Value" />
41:        </event>
42:</public>
43:
44:<script language="VBScript">
45:<![CDATA[
46:
47:Option Explicit
48:
49:Dim strNamingContextSearch
50:Dim strFilterSearch
51:Dim strAttribsToReturnSearch
52:Dim strDepthSearch
53:
54:Function Put_strNamingContextSearch (newValue)
55:         strNamingContextSearch = newValue
56:End Function
57:
58:Function Put_strFilterSearch (newValue)
59:         strFilterSearch = newValue
60:End Function
61:
62:Function Put_strAttribsToReturnSearch (newValue)
63:         strAttribsToReturnSearch = newValue
64:End Function
65:
66:Function Put_strDepthSearch (newValue)
```

```
 67:         strDepthSearch = newValue
 68:End Function
 69:
 70:Function Execute()
 ..:
 87:        Set objDictionary = CreateObject ("Scripting.Dictionary")
 88:
 89:        Set objADOConnnection = CreateObject("ADODB.Connection")
 90:        objADOConnnection.Provider = "ADsDSOObject"
 91:        objADOConnnection.Open "Active Directory Provider"
 92:
 93:        Set objCommand = CreateObject("ADODB.Command")
 94:        Set objCommand.ActiveConnection = objADOConnnection
 95:
 96:        If Ucase (Mid (strNamingContextSearch, 1, 7)) = "LDAP://" Or _
 97:           Ucase (Mid (strNamingContextSearch, 1, 5)) = "GC://" Then
 98:
 99:           Set objNamingContext = GetObject(strNamingContextSearch)
100:        Else
101:           Set objRoot = GetObject("LDAP://RootDSE")
102:           strNamingContext = objRoot.Get(strNamingContextSearch)
103:           Set objRoot = Nothing
...:
109:           Set objNamingContext = GetObject("LDAP://" & strNamingContext)
110:        End If
111:
112:        strADsPathSearch = "<" & objNamingContext.ADsPath & ">"
113:
114:        objCommand.CommandText = strADsPathSearch & ";" & _
115:                                 strFilterSearch & ";" & _
116:                                 strAttribsToReturnSearch & ";" & _
117:                                 strDepthSearch
118:
119:        Set objRecordSet = objCommand.Execute
120:
121:        objDictionary.Add "RecordCount", objRecordSet.RecordCount
122:
123:        While Not objRecordSet.EOF
124:            For intIndice = 0 To objRecordSet.Fields.Count - 1
125:                objDictionary.Add objRecordSet.Fields(intIndice).Name & ":" & _
126:                                  objRecordSet.AbsolutePosition & intIndice, _
127:                                  objRecordSet.Fields(intIndice).Value
128:                FireEvent "Debug", objRecordSet.Fields(intIndice).Name & ":" & _
129:                                   objRecordSet.AbsolutePosition & intIndice & "=" & _
130:                                   objRecordSet.Fields(intIndice).Value
131:            Next
132:
133:            objRecordSet.MoveNext
134:        Wend
135:
136:        Set Execute = objDictionary
137:
138:        Set objNamingContext = Nothing
139:        Set objCommand = Nothing
140:        Set objADOConnnection = Nothing
141:        Set objDictionary = Nothing
142:
143:End Function
144:
145:]]>
146:</script>
147:
148:</component>
```

Sample 9 Windows Script File capturing events

```
 1:' VB Script to show how to use the ADSearch function from a Windows Script      '
 2:' component. (With event sinking)                                               '
 3:'                                                                               '
 4:' Version 1.00 - Alain Lissoir                                                  '
 5:' Compaq Computer Corporation - Professional Services - Belgium -               '
 6:'                                                                               '
 7:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 8:
 9:Option Explicit
10:
11:Dim objADSearch
12:Dim objResultList
13:Dim objResult
14:Dim intColumnPosition
15:
16:Set objADSearch = Wscript.CreateObject("ADSearch.WSC", "ADSearch_")
17:
18:objADSearch.NamingContext = "DefaultNamingContext"
19:objADSearch.Filter = "(|(objectClass=domainDNS)(objectClass=organizationalUnit))"
20:objADSearch.Attributes = "name,distinguishedName"
21:objADSearch.Depth = "subTree"
22:
23:Set objResultList = objADSearch.Execute
24:
25:WScript.Echo
26:WScript.Echo "Number of record found is " & objResultList.Item ("RecordCount")
27:WScript.Echo "Number of elements in the Scripting Dictionary object is " & objResultList.Count
28:
29:For Each objResult in objResultList
30:    intColumnPosition = InStr (objResult, ":")
31:    If intColumnPosition Then
32:       Wscript.Echo Mid (objResult, 1, intColumnPosition - 1) & _
33:                    "=>" & objResultList.Item (objResult)
34:    End If
35:Next
36:
37:' ------------------------------------------------------------------------------------------
38:' Event Handler
39:Sub ADSearch_Debug(Value)
40:    WScript.Echo "Debug: " & Value
41:End Sub
```

Using the ADSearch script to get more Schema information

In the first part of the study "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000", an Active Directory context was loaded in an Excel sheet. As explained, some supplemental characteristic of Active Directory objects attributes called the **systemFlags** was needed. Before going further, some terms need to be clarified:

attributeSchema: An **attributeSchema** contains the definition of an attribute. This definition is composed of several other attributes. It is located in the Active Directory schema.

systemFlags: The **systemFlags** attribute contains a flag enumeration defining:

- Domain-replicated, stored attributes
- Non-replicated, locally stored attributes
- Non-stored, constructed attributes

The **systemFlags** is an attribute of any **attributeSchema.**

LDAPDisplayName: The **lDAPDisplayName** attribute is the name visible when browsing object attributes with the ADSIEDIT MMC (ADSIEDIT is available in the Support Tools on the Windows 2000 CD). The **lDAPDisplayName** is also an attribute of any **attributeSchema**.

The reusable ADSearch function will facilitate the search of the **attributeSchema** for a given object attribute. All object attributes are defined as an **attributeSchema** with a specific name. The **lDAPDisplayName** is different from the name used for

the **attributeSchema**. The problem resides in the way the **lDAPDisplayName** is linked with the name used in the **attributeSchema.**

The **attributeSchema** is unique in the Active Directory. There is a rule to determine the **lDAPDisplayName** from the **attributeSchema** name. (See Active Directory Service Interfaces SDK documentation for more details.) As consequence, the **lDAPDisplayName** is unique too.

From that standpoint, a query of **OneLevel** search to retrieve the **attributeSchema ADsPath** can be made. The **attributeSchema** having this known **lDAPDisplayName** attribute will be returned in the query result. Because the **attributeSchema** common name is unique in the directory (and used to create the **lDAPDisplayName** based on a rule), there is no risk of receiving several matches for the query.

For the ADSearch reuse, the new script will use the XML encapsulation technique (as shown in Sample 3 on page 14).

The ADSearch function can return more than one attribute from the query, so why return **ADsPath** only (see Sample 10 at line 240) instead of returning **systemFlags**? It is possible to return the needed attribute immediately, but having **ADsPath** allows the calling module to bind to the object and perform any other operations on it as well.

To illustrate this, Sample 10 below retrieves the **systemFlags** after a bind operation to the **attributeSchema**. By doing so, we can answer some other interesting questions, such as:

- Is this attribute copied to the Global Catalog?
- Is this attribute indexed?

Of course, again, it is possible to retrieve these values from the query, but in order to vary the samples; this case retrieves these values via a bind operation on the retrieved **ADsPath**.

Sample 10: Loading all the Active Directory Tree objects with their attributes, attribute particularities and syntaxes into an Excel sheet.

```
  1:<!-- VB Script loading all objects from an AD context location into an Excel sheet -->
  2:<!-- (with attributes, syntaxes and system Flags deciphering)                       -->
  3:<!--                                                                               -->
  4:<!-- Version 1.00 - Alain Lissoir                                                  -->
  5:<!-- Compaq Computer Corporation - Professional Services - Belgium -               -->
  6:<!--                                                                               -->
  7:<!-- Any comments or questions:                    EMail:alain.lissoir@compaq.com -->
  8:
  9:<Package>
 10:<job>
 11:        <script language="VBScript" src="Include.vbs" />
 12:
 13:        <script language="VBScript" src="ADSearchFunction.vbs" />
 14:        <script language="VBScript" src="PromptParametersFunction.vbs" />
 15:
 16:        <script language="VBScript">
 ..:
 ..:        < NO CHANGE IN THE CORE COMPONENT, SEE PART 1>
 ..:
 85:        ' ------------------------------------------------------------------------------------
 86:        Private Sub LookInsideObject (strObject, intX)
...:
...:        < NO CHANGE IN THE FUNCTION, SEE PART 1 >
...:
105:        End Sub
106:
107:        ' ------------------------------------------------------------------------------------
108:        Private Sub GetMemberInfo (objObject, objObjectClass, intX)
...:
...:        < NO CHANGE IN THE FUNCTION, SEE PART 1 >
...:
122:        End Sub
```

```
123:
124:        ' ------------------------------------------------------------------------------------
125:        Private Sub LoadPropertiesInXL (PropertyList, objObjectClass, intX)
...:
139:                For Each strProperty in PropertyList
140:                    strADsPathAttributeSchema = ADsPathAttributeSchema (strProperty)
141:                    Set objPropertySchema = GetObject (strADsPathAttributeSchema)
142:
143:                    varSearchFlags = objPropertySchema.Get ("searchFlags")
144:                    If Err.Number = &h8000500D Then
145:                       varSearchFlags = ""
146:                       Err.Clear
147:                     End If
148:
149:                    varSystemFlags = objPropertySchema.Get ("systemFlags")
150:                    If Err.Number = &h8000500D Then
151:                       varSystemFlags = ""
152:                       Err.Clear
153:                    End If
154:
155:                    boolGCCopy = objPropertySchema.Get ("isMemberOfPartialAttributeSet")
156:                    If Err.Number = &h8000500D Then
157:                       boolGCCopy = False
158:                       Err.Clear
159:                    End If
160:
161:                    Set objProperty = GetObject(objObjectClass.Parent + "/" + strProperty)
162:
163:                    intY = intY + 1
164:                    objXL.activecell.offset(intY, intX).Value = objProperty.Name
165:                    objXL.activecell.offset(intY, intX + 1).Value = objProperty.Syntax
166:                    objXL.activecell.offset(intY, intX + 2).Value = objProperty.Multivalued
167:
168:                    If (varSearchFlags And 1) Then
169:                       objXL.activecell.offset(intY, intX + 3).Value = "Indexed"
170:                    Else
171:                       objXL.activecell.offset(intY, intX + 3).Value = ""
172:                    End If
173:
174:                    If (boolGCCopy) Then
175:                       objXL.activecell.offset(intY, intX + 4).Value = "GC Copy"
176:                    Else
177:                       objXL.activecell.offset(intY, intX + 4).Value = ""
178:                    End If
179:
180:                    If varSystemFlags <> "" Then
181:                       objXL.activecell.offset(intY, intX + 5).Value = SystemFlags (varSystemFlags)
182:                    End If
183:
184:                    Set objProperty = Nothing
185:                    Set objPropertySchema = Nothing
186:                Next
187:
188:        End Sub
189:
190:        ' ------------------------------------------------------------------------------------
191:        Function SystemFlags (varSystemFlags)
192:
193:        Dim strTemp
194:
195:                If (varSystemFlags And ADS_SYSTEMFLAG_DISALLOW_DELETE) Then
196:                   strTemp = strTemp & "ADS_SYSTEMFLAG_DISALLOW_DELETE;"
197:                End If
...:
222:                If (varSystemFlags And ADS_SYSTEMFLAG_ATTR_IS_CONTRUCTED) Then
223:                   strTemp = strTemp & "ADS_SYSTEMFLAG_ATTR_IS_CONTRUCTED;"
224:                End If
225:
```

```
226:                SystemFlags = strTemp
227:
228:        End Function
229:
230:        ' ------------------------------------------------------------------------------------------
231:        Private Function ADsPathAttributeSchema (strPropertyName)
...:
237:        Set objResultList = ADSearch ("SchemaNamingContext", _
238:                                      "(&(objectCategory=attributeSchema)(lDAPDisplayName=" _
239:                                                                        & strPropertyName & "))",_
240:                                      "ADsPath", _
241:                                      "OneLevel", _
242:                                      False)
243:
244:        If objResultList.Item ("RecordCount") = 1 Then
245:           For Each objResult in objResultList
246:               intColumnPosition = InStr (objResult, ":")
247:               If intColumnPosition Then
248:                  ADsPathAttributeSchema = objResultList.Item (objResult)
249:                  Exit Function
250:               End If
251:           Next
252:        Else
253:           ADsPathAttributeSchema = ""
254:        End If
255:
256:        End Function
257:
258:        </script>
259:</job>
260:
261:</Package>
```

The core structure of Sample 10 is the same as that presented in the first part of the study "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000". Some modifications were made from the original code to complete the Excel sheet with the new information (**systemFlags**, **searchFlags**, **isMemberOfPartialAttributeSet**).

Line 11 includes the 'Include.vbs' file defining the **systemFlags** constants to interpret its value. This will be used in the new function called 'SystemFlags' (line 191). The functions 'LookInsideObject' (line 86) and 'GetMemberInfo' (line 108) are exactly the same as in the first part "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000".

The most important change is made to the 'LoadPropertiesInXL' function (line 125). At line 140, the script calls the function 'QueryADsPathAttributeSchema' with the attribute name as parameter. This will return the **ADsPath** to the **attributeSchema** defining that attribute. After, it binds to the **attributeSchema** (line 141 and retrieves the following attributes:

- **searchFlags** (line 143)

  It is an integer value containing a bit flag that defines whether the attribute is indexed (if the least significant bit is set to 1) or non-indexed (if the bit is zero). See Active Directory programmer's guide for other bits significations.

- **systemFlags** (line 149)

  It is an integer value containing flags that define additional properties of the attribute such as whether the attribute is constructed, whether it is non-replicated, and whether it is category 1 or 2.

- **isMemberOfPartialAttributeSet** (line 155)

  It is a Boolean value that defines whether the attribute is replicated to the global catalog (it is in the global catalog if it has a value of True and not in the global catalog if the value is False).

Note: Restrictions are imposed on modification of existing schema objects. The schema is categorized into two categories. The schema objects that ship with Windows 2000 in the base schema belong to Category 1. Any schema objects added

later by other applications or users through dynamic schema extension belong to Category 2. Objects of Category 1 can be altered in a very limited way. See Active Directory Service Interfaces SDK for more details about object Category.

On lines 168, 174, 180, some formatting operations are executed based on the value retrieved from the Active Directory. The purpose is to have a nice display output in the Excel Sheet. For this, the 'SystemFlags' function (line 191) decomposes the flag values.

The 'QueryADsPathAttributeSchema' function is nothing other than the standard function call to the ADSearch function. The main difference is the query formulation (lines 237 to 242). First of all, the query is concentrated in the Active Directory Schema context; next, the query string is built to filter only the **attributeSchema** having the **lDAPDDisplayName** attribute equal to the attribute examined in the original 'LoadPropertiesInXL' function. As result of the query, the **ADsPath** is returned. Because the search is done through the schema, the **OneLevel** search is used.

```
237:        Set objResultList = ADSearch ("SchemaNamingContext", _
238:                                      "(&(objectCategory=attributeSchema)(lDAPDisplayName=" _
239:                                                                & strPropertyName & "))", _
240:                                      "ADsPath", _
241:                                      "OneLevel", _
242:                                      False)
```

To complete the information, Table 1 on page 80 shows the worksheet output by the script when when a user object distinguished name is entered.

Note: Sample 10 on page 29 includes a new function called 'PromptParametersFunction.vbs' (line 14). This function is a simple helper to prompt the user for the desired parameters. The code is presented below.

Sample 11 Prompting for some parameters

```
  1:' VB Script to prompt the user for parameters                                  '
  2:'                                                                              '
  3:' Version 1.00 - Alain Lissoir                                                 '
  4:' Compaq Computer Corporation - Professional Services - Belgium -              '
  5:'                                                                              '
  6:' Any comments or questions:                  EMail:alain.lissoir@compaq.com '
  7:
  8:Option Explicit
  9:
 10:' ------------------------------------------------------------------------------------------
 11:Private Function PromptBasicParameters (strTitle, strPrompt, strInitial)
 12:
 13:Dim strInput
 14:
 15:        strInput = InputBox (strTitle, strPrompt, strInitial)
 16:
 17:        If strInput = "" Then
 18:           Wscript.Echo "Sorry, but you didn't enter anything !"
 19:           WScript.Quit (0)
 20:        Else
 21:           PromptBasicParameters = strInput
 22:        End If
 23:
 24:End Function
```

Accessing security settings on File System and Active Directory Objects

Accessing security settings of Active Directory objects requires the use of a new kind of object. Each object in the Active Directory has an attribute called **nTSecurityDescriptor.** The **nTSecurityDescriptor** contains what it is called a 'Security Descriptor' pointer.

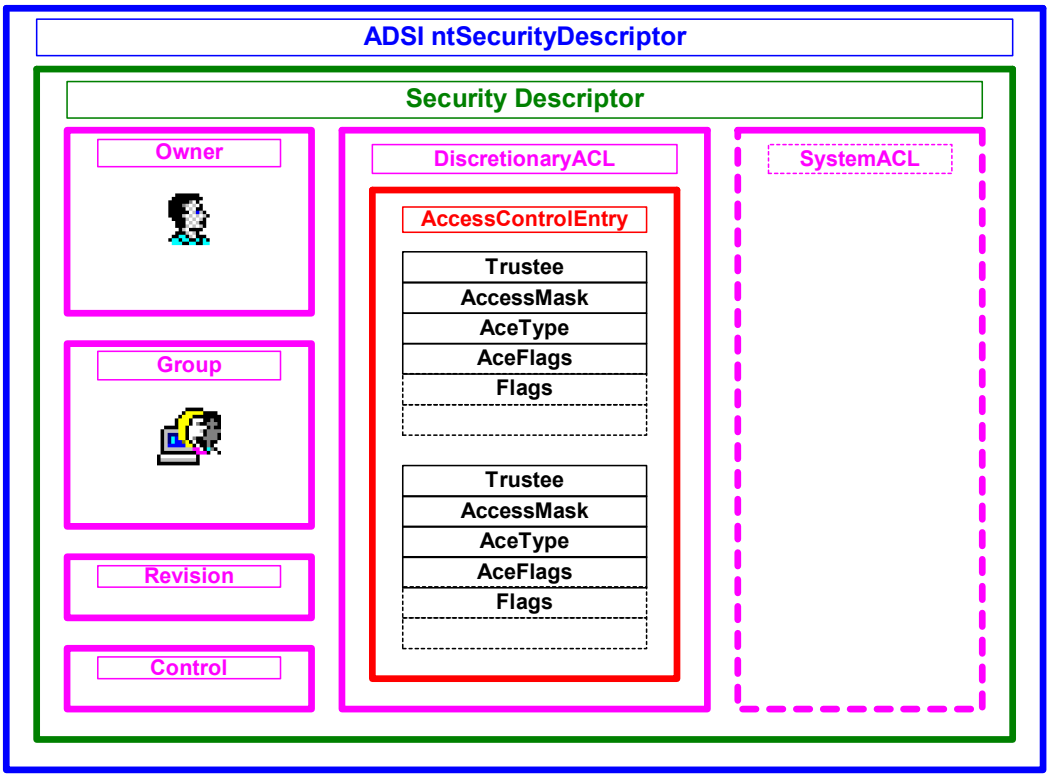


Figure 3 The security descriptor and the relationship between each of its components.

The security descriptor has four particular properties:

- **Owner:** It identifies the owner of the security descriptor.
- **Group:** It identifies the group of the security descriptor.
- **Revision:** Specifies the ACL's revision level. All ACEs in an ACL must be at the same revision level.
- **Control:** It is a set of bit flags that describe a security descriptor and/or its components.

The ACLs are a kind of container holding Access –Control –Entries (ACEs). ACEs are objects that define, on a per user basis, what the user is authorized to do. This is the smallest data element really defining the user privileges and permissions of an object.

There are two kinds of ACLs: the Discretionary ACL (DACL) and the System ACL (SACL). The DACL handles access control on objects. The SACL handles the system auditing on objects. This paper concentrates on DACL because it is the security entity granting privileges and permissions.

To define each privilege and permission for a given user, an ACE has several entries (properties). An ACE is composed of:

- **Trustee:** A trustee is a user, a group or a computer with access rights to an object (i.e. DomainName\UserName).
- **AccessMask:** The AccessMask contains the values assigned from an enumeration list called ADS_RIGHTS_ENUM. This **AccessMask** is a sequence of bits turned on or off to set or remove a specific right.
- **AceType:** The value assigned to this ACE entry determines the type of permissions granted for the corresponding **AccessMask**. (Allowed, Denied, Audit) The value entered here comes from the ADS_ACETYPE_ENUM enumeration.
- **AceFlags:** This entry contains the inheritance control flags. The value entered here comes from the ADS_ACEFLAG_ENUM enumeration.
- **Flags:** The flags value can be used to indicate the presence of the **ObjectType** or **InheritedObjectType** fields in the ACE. The value entered here comes from the ADS_FLAGTYPE_ENUM enumeration.
- **ObjectType:** It indicates what object type, property set, or property an ACE refers to. It takes a GUID as a value.
- **InheritedObjectType:** It specifies the GUID of an object that will inherit the ACE.

Note: For more information about Windows 2000 security architecture, please refer to the White Paper by Jan De Clercq called 'Security Fundamentals as implemented in Windows 2000'.

Security descriptors are objects coming from the Windows NT System. The security descriptor structure is not defined in the Active Directory Schema. Only the attributes containing a security descriptor pointer are defined in the Active Directory Schema. This means that the structural view of the security descriptor is the same for any object in a Windows NT System (Active Directory or File System objects). ADSI only provides objects representing the security descriptor (and methods to retrieve different elements that are part of a security descriptor) coming from an Active Directory (or Exchange Directory).

To access the security descriptor on an object from the file system (a folder or a file), ADSI is not directly helpful because it does not provide methods to read the security descriptor from the file system. A file system 'Security Descriptor' is not stored in an Active Directory attribute; it is stored in the file system itself.

To solve that problem, a specific .DLL provides, as a COM object, different properties and methods to get a security descriptor from "any" object type. This .DLL must be registered in the system. This .DLL is called ADsSecurity.DLL.

RegSvr32.Exe ADsSecurity.DLL

Note: This COM extension is not dedicated to Active Directory and Windows 2000. Sample 12 will run properly under Windows NT 4.0 if WSH 2.0 is installed, if the ADsSecurity.DLL is registered and if a File System object is accessed.

Warning: This .DLL is part of the Active Directory Service Interfaces SDK. This component is unsupported by Microsoft and is not part of the released ADSI that is included with Windows 2000.

Viewing the rights set on a File System Object or on an Active Directory object

It is time to examine how to decipher the security descriptors associated with an object coming from the File System or from the Active Directory. The scope of the security descriptor examination is limited to the standard rights. Active Directory Extended-Rights can also be examined via the same technique but requires a more complex algorithm and is outside of the scope of this document.

It is important to understand how to decipher a security descriptor (Sample 12) because this will be the basis for examining the Add/Remove rights functions later. Sample 12 is written to support security descriptor access for objects coming from the File System or from the Active Directory. Once the security descriptor pointer is retrieved, the process for deciphering the descriptor is always the same.

First of all, the different ENUM tables are defined in the 'Include.vbs' in order to identify the different flag values. It is the calling script's responsibility to make this inclusion. These values come from the Windows NT SDK include files.

Note: To have a complete understanding of all the flags defined, please refer to the Active Directory Service Interfaces SDK documentation and to the Windows 2000 SDK documentation.

To understand the relationship between the effective rights set via the GUI and existing flags, please refer to the Table 2 on page 90 and Table 3 on page 91.

Sample 12: Deciphering a security descriptor from the File System or from the Active Directory.

```
 1:' VB Script to view ACE on File System and Active Directory objects           '
 2:'                                                                             '
 3:' Version 1.00 - Alain Lissoir                                                '
 4:' Compaq Computer Corporation - Professional Services - Belgium -             '
 5:'                                                                             '
 6:' Any comments or questions:                  EMail:alain.lissoir@compaq.com '
 7:
 8:Option Explicit
 9:
10:' -------------------------------------------------------------------------------------------
11:Public Sub ViewRights (strObject, strObjectType)
..:
17:        Select Case strObjectType
18:               Case FS_OBJECT
19:                    Set objSecurity = CreateObject("ADsSecurity")
20:                    Set objSD = objSecurity.GetSecurityDescriptor("FILE://" & strObject)
21:                    WScript.DisconnectObject objSecurity
22:                    Set objSecurity = Nothing
23:               Case AD_OBJECT
24:                    Set objADObject = GetObject("LDAP://" & strObject)
25:                    Set objSD = objADObject.Get("ntSecurityDescriptor")
26:                    WScript.DisconnectObject objADObject
27:                    Set objADObject = Nothing
28:               Case Else
29:                    Exit Sub
30:        End Select
31:
32:        WScript.Echo "Deciphering Security Descriptor " & _
33:                     "'ntSecurityDescriptor'" & vbCRLF & _
34:                     "of Object '" & strObject & "'" & vbCRLF
35:        Call Decipher_SD (strObjectType, objSD)
..:
40:End Sub
41:
42:' -------------------------------------------------------------------------------------------
43:Private Sub Decipher_SD (strObjectType, objSD)
..:
47:        ' -----------------------------------------------------------------------------------
48:        ' Security Descriptor
49:        WScript.Echo "Security Descriptor Owner: '" & objSD.Owner & "'"
50:        WScript.Echo "Security Descriptor Group: '" & objSD.Group & "'"
51:        WScript.Echo "Security Descriptor Revision: '" & objSD.Revision & "'"
52:
53:        '---- SECURITY_DESCRIPTOR_CONTROL ----
54:        WScript.Echo "Security Descriptor Control: '0x" & Hex(objSD.Control) & "'"
55:
56:        If (objSD.Control And SE_OWNER_DEFAULTED) Then
```

```
  57:           WScript.Echo "-SE_OWNER_DEFAULTED"
  58:        End If
  ..:
  92:        If (objSD.Control And SE_SELF_RELATIVE) Then
  93:           WScript.Echo "-SE_SELF_RELATIVE"
  94:        End If
  95:
  96:        WScript.Echo "Security Descriptor OwnerDefaulted: '" & objSD.OwnerDefaulted & "'"
  97:        WScript.Echo "Security Descriptor GroupDefaulted: '" & objSD.GroupDefaulted & "'"
  98:        WScript.Echo "Security Descriptor DaclDefaulted: '" & objSD.DaclDefaulted & "'"
  99:        WScript.Echo "Security Descriptor SaclDefaulted: '" & objSD.SaclDefaulted & "'" & vbCRLF
 100:
 101:        ' ------------------------------------------------------------------------------------
 102:        ' Discretionary ACL
 103:        Set objDACL = objSD.DiscretionaryAcl
 104:
 105:        Select Case strObjectType
 106:               Case FS_OBJECT
 107:                    Call DecipherFS_ACE (objDACL)
 108:               Case AD_OBJECT
 109:                    Call DecipherAD_ACE (objDACL)
 110:        End Select
 111:
 112:        ' Show the number of trustees present in DACL
 113:        WScript.Echo "ACL Revision: " & objDACL.ACLRevision
 114:        WScript.Echo "ACE Count: " & objDACL.AceCount
 115:        WScript.Echo
 ...:
 120:End Sub
 121:
 122:' ------------------------------------------------------------------------------------------
 123:Private Sub DecipherAD_ACE (objDACL)
 ...:
 127:        For Each objACE In objDACL
 128:            WScript.Echo "Trustee: '" & objACE.Trustee & "'"
 129:            WScript.Echo " ACE Type: '" & objACE.AceType & "'"
 130:
 131:            '---- ADS_ACETYPE_ENUM -----
 132:            Select Case objACE.AceType
 133:                   Case ADS_ACETYPE_ACCESS_ALLOWED
 134:                        WScript.Echo " -ADS_ACETYPE_ACCESS_ALLOWED"
 135:                   Case ADS_ACETYPE_ACCESS_DENIED
 136:                        WScript.Echo " -ADS_ACETYPE_ACCESS_DENIED"
 ...:
 145:                   Case Else
 146:
 147:            End Select
 148:
 149:            '--- ADS_RIGHTS_ENUM ------
 150:            WScript.Echo " Access Mask: '0x" & Hex(objACE.AccessMask) & "'"
 151:
 152:            If (objACE.AccessMask = ADS_RIGHT_GENERIC_READ) Then
 153:               WScript.Echo " -ADS_RIGHT_GENERIC_READ"
 154:            End If
 ...:
 219:            If (objACE.AccessMask And ADS_RIGHT_DS_CONTROL_ACCESS) Then
 220:               WScript.Echo " -ADS_RIGHT_DS_CONTROL_ACCESS"
 221:            End If
 222:
 223:            '--- ACE FLAGS ------
 224:            WScript.Echo " ACE Flags: '" & objACE.AceFlags & "'"
 225:
 226:            If (objACE.AceFlags And OBJECT_INHERIT_ACE) Then
 227:               WScript.Echo " -OBJECT_INHERIT_ACE"
 228:            End If
 ...:
 247:            If (objACE.AceFlags And FAILED_ACCESS_ACE_FLAG) Then
 248:               WScript.Echo " -FAILED_ACCESS_ACE_FLAG"
```

```
249:            End If
250:
251:            '--- ADS_FLAGTYPE_ENUM ------
252:            WScript.Echo " Flag Type: '" & objACE.Flags & "'"
253:
254:            If (objACE.Flags And ADS_FLAG_OBJECT_TYPE_PRESENT) Then
255:               WScript.Echo " -ADS_FLAG_OBJECT_TYPE_PRESENT"
256:            End If
257:            If (objACE.Flags And ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT) Then
258:               WScript.Echo " -ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT"
259:            End If
260:
261:            WScript.Echo " Object Type: '" & objACE.ObjectType & "'"
262:            WScript.Echo " Object Inherited Type: '" & objACE.InheritedObjectType & "'" & vbCRLF
263:        Next
264:
265:End Sub
266:
267:' ------------------------------------------------------------------------------------------------
268:Private Sub DecipherFS_ACE (objDACL)
...:
272:        For Each objACE In objDACL
273:            WScript.Echo "Trustee: '" & objACE.Trustee & "'"
274:            WScript.Echo " ACE Type: '" & objACE.AceType & "'"
275:
276:            '---- FILE_ACETYPE_ENUM -----
277:            Select Case objACE.AceType
278:                   Case ACCESS_ALLOWED_ACE_TYPE
279:                        WScript.Echo " -ACCESS_ALLOWED_ACE_TYPE"
280:                   Case ACCESS_DENIED_ACE_TYPE
281:                        WScript.Echo " -ACCESS_DENIED_ACE_TYPE"
282:                   Case SYSTEM_AUDIT_ACE_TYPE
283:                        WScript.Echo " -SYSTEM_AUDIT_ACE_TYPE"
284:                   Case SYSTEM_ALARM_ACE_TYPE
285:                        WScript.Echo " -SYSTEM_ALARM_ACE_TYPE"
286:                   Case Else
287:
288:            End Select
289:
290:            '--- FILE_RIGHTS_ENUM ------
291:            WScript.Echo " Access Mask: '0x" & Hex(objACE.AccessMask) & "'"
292:
293:            If (objACE.AccessMask = FILE_ALL_ACCESS) Then
294:               WScript.Echo " -FILE_ALL_ACCESS"
295:            End If
...:
357:            If (objACE.AccessMask And FILE_SYNCHRONIZE) Then
358:               WScript.Echo " -FILE_SYNCHRONIZE"
359:            End If
360:
361:            '--- ACE FLAGS ------
362:            WScript.Echo " ACE Flags: '" & objACE.AceFlags & "'"
363:
364:            If (objACE.AceFlags And OBJECT_INHERIT_ACE) Then
365:               WScript.Echo " -OBJECT_INHERIT_ACE"
366:            End If
...:
385:            If (objACE.AceFlags And FAILED_ACCESS_ACE_FLAG) Then
386:               WScript.Echo " -FAILED_ACCESS_ACE_FLAG"
387:            End If
388:
389:            '--- ADS_FLAGTYPE_ENUM ------
390:            WScript.Echo " Flag Type: '" & objACE.Flags & "'"
391:            WScript.Echo " Object Type: '" & objACE.ObjectType & "'"
392:            WScript.Echo " Object Inherited Type: '" & objACE.InheritedObjectType & "'" & vbCRLF
393:        Next
394:
395:End Sub
```

When an access to the File System object is made to get the security descriptor, an instance of the *ADsSecurity* object (line 19) is created. The ADsSecurity.DLL provides this *ADsSecurity* object. Next, the method 'GetSecurityDescriptor' is invoked to get the File System security descriptor. This will assign the 'objSD' variable (line 20). On the other hand, when an access to the Active Directory is made to read the security descriptor attribute of the selected object, no instance of the *ADsSecurity* object must be created. Instead of using the 'GetSecurityDescriptor' method, the script binds to the object and it invokes a traditional 'Get' method to get the security descriptor pointer (lines 24 and 25). This is because ADSI provides an object representing a security descriptor (such as *AccessControlList* and *AccessControlEntry* object).

Actually to get the security descriptor from the Active Directory, it is possible to use the same method as for the File System access. (Lines 19 to 22) In this case, the lines 24 to 27 could be easily replaced by:

```
Set objSecurity = CreateObject("ADsSecurity")
Set objSD = objSecurity.GetSecurityDescriptor("LDAP://" & strObject)
WScript.DisconnectObject objSecurity
Set objSecurity = Nothing
```

which is functionally equivalent to the original code:

```
Set objADObject = GetObject("LDAP://" & strObject)
Set objSD = objADObject.Get("ntSecurityDescriptor")
WScript.DisconnectObject objADObject
Set objADObject = Nothing
```

Note that instead of using a "FILE://" pointer, an "LDAP://" pointer is used; this is the only real difference. The 'GetSecurityDescriptor' method accepts both pointers. Why use two different methods: one for the File System security descriptor retrieval and another one for the security descriptor located in the Active Directory?

The **nTSecurityDescriptor** attribute is not the only attribute on a user object in the Active Directory containing a security descriptor pointer. When Exchange 2000 is installed under Windows 2000, there are a lot of schema modifications. One of these modifications is the addition of a new attribute associated with the user object: **msExchMailboxSecurityDescriptor**. The problem with invoking the 'GetSecurityDescriptor' method is that it returns only the **nTSecurityDescriptor** attribute. There is no way (up to now) to get the **msExchMailboxSecurityDescriptor** when invoking that method. So, using the general 'Get' method allows reading the **ntSecurityDescriptor** or **msExchMailboxSecurityDescriptor**. In fact, this is a more general method. It will be useful later.

Note: It is important to know that the ADSI SDK 2.5 provides the ADsSecurity.DLL. This ADSI SDK is not part of Windows 2000 or Exchange 2000. It is possible that a new or updated version of ADsSecurity.DLL (or an equivalent solution) will be provided with future releases to support an access method to the **msExchMailboxSecurityDescriptor** attribute.

Unfortunately, to decipher the **msExchMailboxSecurityDescriptor**, the flag values are not the same as flag values used to decipher **ntSecurityDescriptor.** Even if the method to get the specific security descriptor is the same, the flag values and their meaning are different. It will be explained later in this document how to build a security descriptor for Exchange 2000 (see Sample 17 on page 49).

Once the security descriptor is assigned to the 'objSD' variable (lines 20 and/or 25), it is possible to decompose the security descriptor to each of its components:

- .Owner property to get the security descriptor owner (line 49).
- .Group property to get the security descriptor group (line 50).
- .Revision property to get the security descriptor revision (line 51).
- .Control property to get the security descriptor control flags (line 54).
- .DiscretionaryAcl method to extract the Discretionary ACL (line 103).

From lines 56 to 94, the Control flag is deciphered to identify the SECURITY_DESCRIPTOR_CONTROL flag values. From line 96 to 99, different Boolean values are retrieved from properties associated with the security descriptor object:

- .OwnerDefaulted
- .GroupDefaulted
- .DaclDefaulted
- .SaclDefaulted

These Boolean values are used during the creation of a security descriptor. They indicate if the information is derived from a default mechanism rather than explicitly set by the original provider of the security descriptor (in other words, the script). These values are not really a property of the security descriptor but they are properties of the security object created at line 20 and/or 25. When running the script, these flags will be always False.

From line 105, the Discretionary ACL is deciphered down to each of its components. This operation is performed differently based on the type of object examined: a File System object (line 107) or an Active Directory object (line 109). The deciphering method is not really different between a Discretionary ACL coming from the File System and a Discretionary ACL coming from the Active Directory. This operation is divided into two different functions because flag values and flag meanings are not the same. This will give a coherent output regardless of the object type analyzed.

The extraction (at line 107 for the 'DecipherFS_ACE' function and at line 109 for the 'DecipherAD_ACE' function) of each ACE contained in the Discretionary ACL is done by a simple enumeration. This allows decomposing each ACE present in the Discretionary ACL. For the 'DecipherAD_ACE' function:

- .Trustee is shown at line 128.
- .AceType is shown at line 129 and deciphered from lines 132 to 147.
- .AccessMask is shown at line 150 and deciphered from lines 152 to 221.
- .AceFlags is shown at line 224 and deciphered from lines 226 to 249.
- .Flags is shown at line 252 and deciphered from lines 254 to 259.
- .ObjectType, is shown at lines 261.
- .InheritedObjectType is shown at lines 262.

This is exactly the same principle for the 'DecipherFS_ACE' function.

The **Flags** property determines if the **ObjectType** or **InheritedObjectType** properties are set:

- **ObjectType** indicates what object type, property set, or property the ACE refers to. It takes a GUID as its value. The GUID referenced by **ObjectType** is not physically present in the ACE unless ADS_FLAGS_OBJECT_TYPE_PRESENT is set (see ADS_FLAGTYPE_ENUM). This property plays the role of a pointer (the GUID number) to the Extended Rights set located in the Extended Right container in the configuration context.
- **InheritedObjectType** specifies the schemaIDGUID of an object that will inherit the ACE. The GUID is not physically present in the ACE unless the ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT bit is set.

Note: For more information about Windows 2000 security inheritance and delegation, please refer to the White Paper by Jan De Clercq called 'Security Fundamentals as implemented in Windows 2000'.

Finally, on lines 113 and 114, the script shows two properties from the Discretionary ACL.

- **.ACLRevision:** The revision level of an access-control list. All ACEs in an ACL must be at the same revision level.
- **.AceCount:** The number of access control entries in the access-control list.

To conclude, here is sample code using the ViewRightFunction.VBS.

The script reusability principle is always the same. The core is included in a function and a Windows Script File includes that function in an XML frame.

Sample 13: Using the 'ViewSecurityDescriptor.VBS' script from a Windows Script file.

```
   1:<!-- Windows Script File viewing rights on File System and Active Directory -->
   2:<!-- objects                                                                 -->
   3:<!--                                                                        -->
   4:<!-- Version 1.00 - Alain Lissoir                                           -->
   5:<!-- Compaq Computer Corporation - Professional Services - Belgium -        -->
   6:<!--                                                                        -->
   7:<!-- Any comments or questions:             EMail:alain.lissoir@compaq.com -->
   8:
   9:<job>
  10:        <script language="VBScript" src="Include.vbs" />
  11:        <script language="VBScript" src="ViewRightFunction.vbs" />
  12:
  13:        <script language="VBScript">
  ..:
  20:                WScript.Echo "********** Accessing File System through a remote share **********"
  21:                Call ViewRights("\\MYW2KDC\HOME$\JAMES.WEST", FS_OBJECT)
  22:                WScript.Echo
  23:
  24:                WScript.Echo "******************** Accessing Active Directory ******************"
  25:                Set objRoot = GetObject ("LDAP://RootDSE")
  26:                strNamingContext = objRoot.Get ("defaultNamingContext")
  27:                Call ViewRights("CN=WEST James,CN=Users," & strNamingContext, AD_OBJECT)
  28:                WScript.Echo
  29:
  30:        </script>
  31:</job>
```

Here is the partial display output of an object access to the File System first and to the Active Directory next. Note that the security descriptor for the File System object is accessed through a remote share (UNC).

```
********** Accessing File System through a remote share **********
Deciphering Security Descriptor 'ntSecurityDescriptor'
of Object '\\MYW2KDC\HOME$\LissoirA'

Security Descriptor Owner: 'BUILTIN\Administrators'
Security Descriptor Group: 'W2K-HOME\Domain Users'
Security Descriptor Revision: '1'
Security Descriptor Control: '0x9404'
-SE_DACL_PRESENT
-............
-SE_SELF_RELATIVE
Security Descriptor OwnerDefaulted: 'False'
Security Descriptor GroupDefaulted: 'False'
Security Descriptor DaclDefaulted: 'False'
Security Descriptor SaclDefaulted: 'False'

Trustee: 'W2K-HOME\LissoirA'
 ACE Type: '0'
 -ACCESS_ALLOWED_ACE_TYPE
 Access Mask: '0x1301BF'
 -FILE_READ_DATA
 -............
 -READ_CONTROL
 -SYNCHRONIZE
 ACE Flags: '3'
 -OBJECT_INHERIT_ACE
 -CONTAINER_INHERIT_ACE
 -VALID_INHERIT_FLAGS
 Flag Type: '0'
 Object Type: ''
 Object Inherited Type: ''

 ......................

ACL Revision: 2
```

```
ACE Count: 2

******************** Accessing Active Directory ******************
Deciphering Security Descriptor 'ntSecurityDescriptor'
of Object 'CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com'

Security Descriptor Owner: 'W2K-HOME\Domain Admins'
Security Descriptor Group: 'W2K-HOME\Domain Admins'
Security Descriptor Revision: '1'
Security Descriptor Control: '0x9C04'
-SE_DACL_PRESENT
-............
-SE_SELF_RELATIVE
Security Descriptor OwnerDefaulted: 'False'
Security Descriptor GroupDefaulted: 'False'
Security Descriptor DaclDefaulted: 'False'
Security Descriptor SaclDefaulted: 'False'

Trustee: 'NT AUTHORITY\Authenticated Users'
 ACE Type: '0'
 -ADS_ACETYPE_ACCESS_ALLOWED
 Access Mask: '0x20094'
 -ADS_RIGHT_READ_CONTROL
 -............
 -ADS_RIGHT_DS_LIST_OBJECT
 ACE Flags: '0'
 Flag Type: '0'
 Object Type: ''
 Object Inherited Type: ''

.......................

Trustee: 'W2K-HOME\Domain Admins'
 ACE Type: '0'
 -ADS_ACETYPE_ACCESS_ALLOWED
 Access Mask: '0xE01BF'
 -ADS_RIGHT_READ_CONTROL
 -............
 -ADS_RIGHT_DS_CONTROL_ACCESS
 ACE Flags: '0'
 Flag Type: '0'
 Object Type: ''
 Object Inherited Type: ''

Trustee: 'NT AUTHORITY\SYSTEM'
 ACE Type: '0'
 -ADS_ACETYPE_ACCESS_ALLOWED
 Access Mask: '0xF01FF'
 -ADS_RIGHT_DELETE
 -............
 -ADS_RIGHT_DS_CONTROL_ACCESS
 ACE Flags: '0'
 Flag Type: '0'
 Object Type: ''
 Object Inherited Type: ''
ACL Revision: 2
ACE Count: 5
```

Adding rights to a File System Object or to an Active Directory object

Considering the Active Directory Users and Computers MMC interface showing permissions on an Active Directory object (only available by selecting the advanced view mode), there are many different rights you can set. For each, there is a particular set of flags.

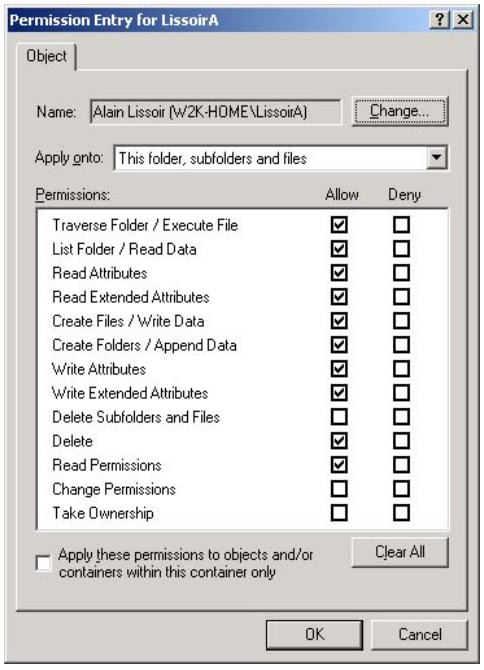


Figure 4 Advanced view of File System rights**.**

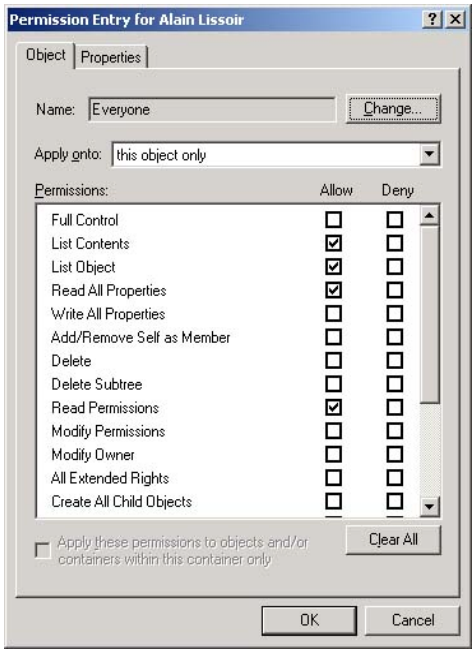


Figure 5 Advanced view of Active Directory rights**.**

A good exercise with Sample 12 on page 35 is to set different rights on a particular object and to discover which flag is activated in respect of the GUI right settings selected. The summary of the standard rights (and inheritance) can be found in Table 2 to Table 5 on page 90.

Note: Extended rights are more complex to manipulate because the ACE **Flags** property is used as a switch to determine the presence of a GUID pointer to the Configuration Context. The **ObjectType** property of the ACE contains this GUID if the Flag property is set to ADS_FLAG_OBJECT_TYPE_PRESENT. The Configuration context contains in its Extended rights container all the extended rights created in the Active Directory. The extended rights container is located in the Configuration context because this particular context is replicated across all the Forest; which makes them available everywhere. In order to simplify this document, the examples are limited to the standard rights only.

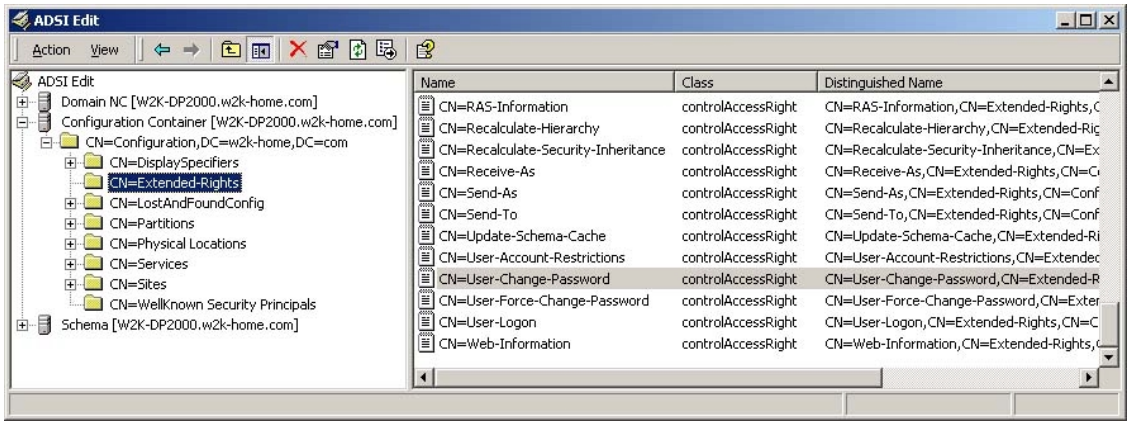


Figure 6 The extended rights location in the Configuration Context.

Setting rights

A right is a combination of different flags. By looking at Table 2 on page 90 for Active Directory rights and Table 3 on page 91 for File System rights, it is easy to determine the flags needed to set a right as shown in the GUI. For example, to set the 'Read' right, the following flags must be used:

- ADS_ACETYPE_ACCESS_ALLOWED to grant the right

or

- ADS_ACETYPE_ACCESS_DENIED to deny to the right.

And then, to explicitly select the 'Read' right, the following flags must be combined:

- ADS_RIGHT_READ_CONTROL
- ADS_RIGHT_ACTRL_DS_LIST
- ADS_RIGHT_DS_READ_PROP
- ADS_RIGHT_DS_LIST_OBJECT

The particularity of the 'Read' right is that the same result can be obtained by using the ADS_RIGHT_GENERIC_READ. After setting an ACE by using the ADS_RIGHT_GENERIC_READ flag, reading back the result will give the flags combination presented before.

The principle is exactly the same for other generic rights (see Table 2 on page 90 for more details):

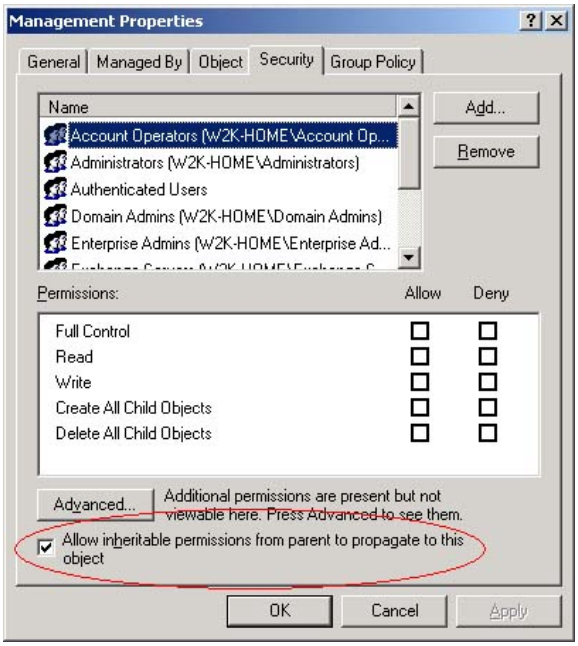
- ADS_RIGHT_GENERIC_WRITE
- ADS_RIGHT_GENERIC_EXECUTE
- ADS_RIGHT_GENERIC_ALL

Note: The generic rights principle is also applicable for File System objects. Please refer to Table 3 on page 91 for a complete list of generic rights.

Inheritance

As rights, inheritance is a flag combination. By looking at Table 4 on page 92 for Active Directory inheritance and at Table 5 on page 92 for File System inheritance, it is easy to determine the flags needed to set an inheritance as shown in the GUI (See Figure 4 and Figure 5 on page 42). For example to set the 'Child objects only' inheritance, the following flags must be used:

- CONTAINER_INHERIT_ACE
- INHERIT_ONLY_ACE



When viewing the rights of an Active Directory object, it is possible to protect the object against inheritance (coming from parent objects). This operation can be completed via the checkbox in the user interface (see Figure 7).

When deciphering a security descriptor, setting the flag value 'SE_DACL_PROTECTED' on the security descriptor object control flag will have the same effect as deselecting the checkbox in the user interface.

This setting is set on the security descriptor object level and not on the ACE or ACL level.

Figure 7 Blocking inheritance

Now the missing part to complete the operation is to have a function adding rights to an object security descriptor. This is the purpose of Sample 14 below.

Sample 14: Adding rights to a File System object or to an Active Directory object.

```
  1:' VB Script to add ACE on File System and Active Directory objects           '
  2:'                                                                            '
  3:' Version 1.00 - Alain Lissoir                                               '
  4:' Compaq Computer Corporation - Professional Services - Belgium -            '
  5:'                                                                            '
  6:' Any comments or questions:                EMail:alain.lissoir@compaq.com '
  7:
  8:Option Explicit
  9:
 10:' ----------------------------------------------------------------------------------------
 11:Public Sub AddRights(strObject,strTrustee,intACEType,intAccessMask,intACEFlags,strObjectType)
 ..:
 20:        Select Case strObjectType
 21:               Case FS_OBJECT
 22:                    Set objSecurity = CreateObject("ADsSecurity")
 23:                    Set objSD = objSecurity.GetSecurityDescriptor("FILE://" & strObject)
 24:               Case AD_OBJECT
 25:                    Set objADObject = GetObject("LDAP://" & strObject)
 26:                    Set objSD = objADObject.Get("ntSecurityDescriptor")
 27:               Case Else
```

```
28:                     Exit Sub
29:         End Select
30:
31:         Set objDACL = objSD.DiscretionaryAcl
32:
33:         Set objNewACE = CreateObject("AccessControlEntry")
34:
35:         ' Add the new Trustee with his specific rights
36:         objNewACE.Trustee = strTrustee
37:         objNewACE.AceType = intACEType
38:         objNewACE.AccessMask = intAccessMask
39:         objNewACE.AceFlags = intACEFlags
40:
41:         objDACL.AddAce objNewACE
42:         objSD.DiscretionaryAcl = objDACL
43:
44:         Select Case strObjectType
45:                Case FS_OBJECT
46:                     objSecurity.SetSecurityDescriptor objSD
47:                     WScript.DisconnectObject objSecurity
48:                     Set objSecurity = Nothing
49:                Case AD_OBJECT
50:                     objADObject.Put "ntSecurityDescriptor", objSD
51:                     objADObject.SetInfo
52:                     WScript.DisconnectObject objADObject
53:                     Set objADObject = Nothing
54:         End Select
55:
56:         Wscript.Echo "  Trustee '" & strTrustee & "' has been added to '" & strObject & "'"
..:
65:End Sub
```

In Sample 14 (Adding rights to a File System object or to an Active Directory object.) on page 44, the script extracts the security descriptor from the given object (line 20 to 29). Next, it extracts the Discretionary ACL (line 31) in order to add the new desired ACE to the ACL. To accomplish this, the script needs to build a new ACE object. That's what it does at line 33. It creates an instance of the 'AccessControlEntry' object.

The next operations are obvious. It assigns the Trustee, the AceType, the AccessMask and AceFlags to the corresponding properties of the new ACE object.

The Discretionary ACL exposes a method to add a new ACE: 'AddAce' (line 41). After this, it puts back the modified Discretionary ACL to the Security Descriptor.

The operation is not complete until it puts back the modified security descriptor to the original object. Here again, as the 'GetSecurityDescriptor' method, if the object is a File System object or an Active Directory object, it uses two different methods. As explained before, it was possible to use the 'SetSecurityDescriptor' method to save the modified security descriptor to the Active Directory. The 'Put' and 'SetInfo' methods combination are used because the 'SetSecurityDescriptor' method has the same limitation as 'GetSecurityDescriptor': These methods only address the **nTSecurityDescriptor** LDAP attribute.

Removing rights from a File System Object or from an Active Directory object

To remove rights, the next script uses the same general structure as Sample 14 above. The difference is in the method used to remove the desired trustee. Because Trustee is a "member" of an ACE and ACEs "members" of the Discretionary ACL, the script makes a 'For Loop' to enumerate all ACEs until it finds the desired one.

Sample 15 uses this method to remove rights from a File System object.

Sample 15: Removing rights from a File System object or from an Active Directory object.

```
 1:' VB Script to remove ACE on File System and Active Directory objects        '
 2:'                                                                            '
 3:' Version 1.00 - Alain Lissoir                                               '
 4:' Compaq Computer Corporation - Professional Services - Belgium -            '
 5:'                                                                            '
 6:' Any comments or questions:                EMail:alain.lissoir@compaq.com '
 7:
 8:Option Explicit
 9:
10:' ------------------------------------------------------------------------------------------
11:Public Sub RemoveRights(strObject, strTrustee, strObjectType)
..:
21:        Select Case strObjectType
22:               Case FS_OBJECT
23:                    Set objSecurity = CreateObject("ADsSecurity")
24:                    Set objSD = objSecurity.GetSecurityDescriptor("FILE://" & strObject)
25:               Case AD_OBJECT
26:                    Set objADObject = GetObject("LDAP://" & strObject)
27:                    Set objSD = objADObject.Get("ntSecurityDescriptor")
28:               Case Else
29:                    Exit Sub
30:        End Select
31:
32:        Set objDACL = objSD.DiscretionaryAcl
33:
34:        For Each objACE In objDACL
35:            Select Case strTrustee
36:                   Case "REMOVE_ALL_RIGHTS"
37:                          objDACL.RemoveAce (objACE)
38:
39:                   Case Else
40:                        If (objACE.Trustee = strTrustee) Then
41:                           Wscript.Echo "  Trustee '" & strTrustee & _
42:                                        "' has been removed from '" & strObject & "'"
43:                           objDACL.RemoveAce (objACE)
44:
45:                           Exit For
46:                        End If
47:            End Select
48:        Next
49:
50:        If strTrustee = "REMOVE_ALL_RIGHTS" Then
51:           Set objNewACE = CreateObject("AccessControlEntry")
52:
53:           ' Add the new Trustee with his specific rights
54:           objNewACE.Trustee = "Everyone"
55:           objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
56:           objNewACE.AccessMask = ADS_RIGHT_GENERIC_ALL
57:           objNewACE.AceFlags = CONTAINER_INHERIT_ACE Or OBJECT_INHERIT_ACE
58:
59:           Wscript.Echo "  All rights are removed, Trustee '" & objNewACE.Trustee & _
```

```
60:                         "' (Full Control) has been added."
61:
62:            objDACL.AddAce objNewACE
..:
67:         End If
68:
69:         objSD.DiscretionaryAcl = objDACL
70:
71:         Select Case strObjectType
72:                Case FS_OBJECT
73:                     objSecurity.SetSecurityDescriptor objSD
..:
76:                Case AD_OBJECT
77:                     objADObject.Put "ntSecurityDescriptor", objSD
78:                     objADObject.SetInfo
..:
81:         End Select
..:
86:End Sub
```

The only difference between Sample 15 above and Sample 14 on page 44 resides on lines 34 to 48. These lines contain the 'For Each' loop making the ACE enumeration until the Trustee you want to remove is found. The next lines contain the usual Discretionary ACL and Security Descriptor saving procedures. Note that it is possible to remove all the rights in one step by assigning the variable 'Trustee' to "REMOVE_ALL_RIGHTS" (lines 36 and 37). For accessibility reasons, after this removal, the script gives the 'Everyone' Full Control right to the same object to be sure that the object is still accessible.

Note: Giving 'Everyone' Full Control is equal to suppressing the concept of controlling the access. From a human point of view, this makes sense but from a computer point of view, removing all access controls means no access. That's why we add this trustee (lines 50 to 67).

Finally, below, this sample code is using the three new functions to view, add and remove rights. Note the four include statements. "Include.vbs" is included first (line 10) in order to have all constant definitions loaded. This sample script adds a right for 'Everyone' to a remote directory (via UNC) on the File System (line 26). Afterwards, the script removes the right just set (line 41). Before and after each operation, the script uses the "ViewRightFunction.VBS" to show the effective rights set on the object.

The script repeats the same operation for an Active Directory Object (lines 48 to 73).

Sample 16: Windows Script demonstrating the miscellaneous right functions usage.

```
 1:<!-- Windows Script File setting right on File System and Active Directory  -->
 2:<!-- objects and showing result between each operation                      -->
 3:<!--                                                                        -->
 4:<!-- Version 1.00 - Alain Lissoir                                          -->
 5:<!-- Compaq Computer Corporation - Professional Services - Belgium -       -->
 6:<!--                                                                        -->
 7:<!-- Any comments or questions:             EMail:alain.lissoir@compaq.com -->
 8:
 9:<job>
10:        <script language="VBScript" src="Include.vbs" />
11:        <script language="VBScript" src="ViewRightFunction.vbs" />
12:        <script language="VBScript" src="AddRightFunction.vbs" />
13:        <script language="VBScript" src="RemoveRightFunction.vbs" />
14:
15:        <script language="VBScript">
..:
22:                WScript.Echo "********** Accessing File System through a remote share ******"
23:                Call ViewRights("\\MYW2KDC\HOME$\JAMES.WEST", FS_OBJECT)
24:                WScript.Echo
25:
26:                Call AddRights("\\MYW2KDC\HOME$\JAMES.WEST", _
27:                               "Everyone", _
```

```
28:                                 ADS_ACETYPE_ACCESS_ALLOWED, _
29:                                 ADS_RIGHT_GENERIC_READ Or _
30:                                 ADS_RIGHT_GENERIC_WRITE Or _
31:                                 ADS_RIGHT_DELETE Or _
32:                                 ADS_RIGHT_GENERIC_EXECUTE, _
33:                                 CONTAINER_INHERIT_ACE Or _
34:                                 OBJECT_INHERIT_ACE, _
35:                                 FS_OBJECT)
..:
38:                 Call ViewRights("\\MYW2KDC\HOME$\JAMES.WEST", FS_OBJECT)
..:
41:                 Call RemoveRights("\\MYW2KDC\HOME$\JAMES.WEST", _
42:                                   "Everyone", _
43:                                   FS_OBJECT)
..:
46:                 Call ViewRights("\\MYW2KDC\HOME$\JAMES.WEST", FS_OBJECT)
47:
48:                 WScript.Echo "******************* Accessing Active Directory ***********"
49:                 Set objRoot = GetObject ("LDAP://RootDSE")
50:                 strObjectADsPath = "CN=WEST James,CN=Users," & _
                                                    objRoot.Get ("defaultNamingContext")
51:                 Call ViewRights(strObjectADsPath, AD_OBJECT)
52:                 WScript.Echo
53:
54:                 Call AddRights(strObjectADsPath, _
55:                                "Everyone", _
56:                                ADS_ACETYPE_ACCESS_ALLOWED, _
57:                                ADS_RIGHT_GENERIC_READ Or _
58:                                ADS_RIGHT_GENERIC_WRITE Or _
59:                                ADS_RIGHT_DELETE Or _
60:                                ADS_RIGHT_GENERIC_EXECUTE, _
61:                                0, _
62:                                AD_OBJECT)
..:
65:                 Call ViewRights(strObjectADsPath, AD_OBJECT)
..:
68:                 Call RemoveRights(strObjectADsPath, _
69:                                   "Everyone", _
70:                                   AD_OBJECT)
..:
73:                 Call ViewRights(strObjectADsPath, AD_OBJECT)
74:
75:         </script>
76:</job>
```

Creating a Security Descriptor for an Exchange 2000 mailbox

The purpose of this section is to show how to create a security descriptor for the **msExchMailboxSecurityDescriptor** user object attribute. The **msExchMailboxSecurityDescriptor** attribute is only present if Exchange 2000 is installed under Windows 2000. The installation of Exchange 2000 makes a huge schema modification to permit E-Mail enabled functions with some Active Directory objects (Group Object, User Object, and so on). In order to access the mailbox, as with any other Active Directory object, a security descriptor must be defined. This is the mailbox access right definition under the Active Directory.

Under Windows NT 4.0, the SAM database is not extensible. This means that the installation of Exchange is not able to modify the SAM database "schema". This forces Microsoft to store mailbox access rights in the "dedicated" Exchange Directory Service. More than setting an access right via a security descriptor, it is necessary to map the primary user mailbox with an NT legacy account (Security ID). Under Exchange 5.5, those two values are visible via LDAP and are as follows:

- **Assoc-NT-Account:** For the primary owner of the mailbox. This attribute contains a Security ID.
- **NT-Security-Descriptor:** For the access rights to the mailbox. This attribute contains a Security Descriptor.

Under Windows 2000, the user object itself contains the mailbox definition. The only difference from a Security Descriptor point of view is that there is a second Security Descriptor definition for the mailbox access right. So, user object has:

- The Security Descriptor defining the access rights to the user object and is stored in a user object attribute called: **ntSecurityDescriptor**.
- The Security Descriptor defining the access rights to the mailbox and is stored in a user object attribute called: **msExchMailboxSecurityDescriptor**.

To summarize, the **msExchMailboxSecurityDescriptor** of Exchange 2000 under Windows 2000 is the equivalent to the **NT-Security-Descriptor** attribute of Exchange 5.5. The **msExchMailboxSecurityDescriptor** uses different flag values than those used by Exchange 5.5 to set the mailbox rights (See Table 6 on page 93).

The purpose of the next script is to build a valid security descriptor for the **msExchMailboxSecurityDescriptor** attribute**.**

Note: When creating Windows 2000 users via LDAP, a default Security Descriptor is built to define the security access rules to the newly created object. On the other hand, the Security Descriptor used for mailbox access rights is not initialized. It must be enabled from the script. This task cannot be delegated to a default mechanism.

Active Directory provides the capability to specify a default security for each type of object. This is specified in the **defaultSecurityDescriptor** attribute of the **classSchema** object definition. This security descriptor is used to provide default protection on the object if there is no security descriptor specified during the creation of the object and if there is no specific inherited permissions from the parent. (See Microsoft Active Directory Service Interfaces 2.5 SDK for more details.)

Sample 17 Creating a security descriptor

```
 1:' VB Script creating Security Descriptor for Active Directory objects and        '
 2:' saving it in the associated Security Descriptor attribute                      '
 3:' (Exchange 2000 mailbox or Active Directory object are supported)               '
 4:'                                                                                '
 5:' Version 1.00 - Alain Lissoir                                                   '
 6:' Compaq Computer Corporation - Professional Services - Belgium -                '
 7:'                                                                                '
 8:' Any comments or questions:                      EMail:alain.lissoir@compaq.com '
 9:
10:Option Explicit
11:
12:' ------------------------------------------------------------------------------------------
13:Public Sub SetSD(varObject, ACE_Entries, strObjectType)
..:
21:        Select Case strObjectType
22:               Case AD_OBJECT, MB_OBJECT
23:
24:               Case Else
25:                    Exit Sub
26:        End Select
```

```
27:
28:         Set objSD = CreateObject("SecurityDescriptor")
29:         Set objDACL = CreateObject("AccessControlList")
30:
31:         objDACL.AceCount = Ubound (ACE_Entries) / 4
32:         objDACL.AclRevision = ADS_ACL_REVISION_DS
33:
34:         For intIndice = 0 To (Ubound (ACE_Entries) - 1) Step 4
35:             ' Create an Access Control Entry (ACE_Entries)
36:             Set objACE = CreateObject("AccessControlEntry")
37:
38:             objACE.Trustee = ACE_Entries (intIndice)
39:             objACE.AceType = ACE_Entries (intIndice + 1)
40:             objACE.AccessMask = ACE_Entries (intIndice + 2)
41:             objACE.AceFlags = ACE_Entries (intIndice + 3)
42:
43:             ' Add the newly created ACE objects to the new access-control list (ACL)
44:             objDACL.AddAce objACE
45:
46:             WScript.DisconnectObject objACE
47:             Set objACE = Nothing
48:         Next
49:
50:         ' Use the created DACL instead of the default.
51:         objSD.Revision = ADS_SD_REVISION_DS
52:         objSD.OwnerDefaulted = True
53:         objSD.GroupDefaulted = True
54:         objSD.DaclDefaulted = False
55:         objSD.SaclDefaulted = True
56:         objSD.DiscretionaryAcl = objDACL
57:
58:         Select Case strObjectType
59:                Case AD_OBJECT
60:                     varObject.Put "nTSecurityDescriptor", objSD
61:                     varObject.SetInfo
62:                     Wscript.Echo "  Security Descriptor for AD object initialized."
63:                Case MB_OBJECT
64:                     varObject.Put "msExchMailboxSecurityDescriptor", objSD
65:                     varObject.SetInfo
66:                     Wscript.Echo "  Security Descriptor for MB object initialized."
67:         End Select
..:
75:End Sub
```

The security descriptor creation follows the same logic used to create an ACE (see Sample 14 on page 44). The principle is to create an instance of the new objects. To create a new ACE, the script has to create an instance of the **AccessControlEntry** object. For a security descriptor, the script must create all object instances constituting a security descriptor. In this case, the instances to create are:

- The **SecurityDescriptor** object (SD) at line 28
- The **AccessControlList** object (ACL) at line 29
- The **AccessControlEntry** object (ACE) at line 36

Creating a security descriptor also means assigning rights. To ease the operation, Sample 17 above accepts an array of ACE entries as a parameter. During the security descriptor initialization, the script will assign the ACE array to each ACE entry of the new Discretionary ACL (line 34 to 48). The number of ACE entries is determined from the array size used to pass the ACE list (line 31). Once a new ACE object instance is created (line 36), the corresponding ACE element is assigned from the array. The script loops until all the ACE array entries are assigned. Each time an ACE from the array is assigned (line 38 to 41), it puts the new ACE into the Discretionary ACL (line 44).

The next piece of code is specific to the security descriptor creation (line 51 to 56). This step defines how the different components constituting a security descriptor are created. For each element of a security descriptor (see Figure 3 on page 33), a flag must be set to define the method used for the creation. All elements are created by a default mechanism except the Discretionary ACL (because the script will set this one). This is the reason why **DaclDefaulted** is set to False (line 54). On line 56, the script sets the new Discretionary ACL to the security descriptor.

The next operation is the usual save process to the Active Directory object having the **nTSecurityDescriptor** (line 60 and 61) or **msExchMailboxSecurityDescriptor** attribute (line 64 and 65).

The array of ACE entries must be organized as follows:

```
Array (Trustee1, ACEType1, AccessMask1, ACEFlags1, _
       Trustee2, ACEType2, AccessMask2, ACEFlags2, _
       ..............).
```

Sample 22 on page 68 shows how to use this function.

Note: Now we are able to create a security descriptor for **msExchMailboxSecurityDescriptor** attribute. An enhancement to be able to manipulate rights for that particular security descriptor is to make the following changes:

- Sample 12 (Deciphering a security descriptor from the File System or from the Active Directory.) on page 35
- Sample 14 (Adding rights to a File System object or to an Active Directory object.) on page 44
- Sample 15 (Removing rights from a File System object or from an Active Directory object.) on page 46

Creating security groups based on OU membership

Organizational units under Active Directory allow creating a container hierarchy to reflect the needs of an organization. The organizational unit is a general-purpose container that can be used to group most other object classes together for purposes of administration. Windows 2000 allows administrators to apply very fine-grained access control to objects in the directory and to delegate administrative authority to other users grouped in the organizational unit.

An organizational unit has a security descriptor to define access rights to the container. It is important to note that an organizational unit is NOT a security principal. This means that no SID is assigned to an organizational unit. The consequence is that an administrator cannot use this kind of object to grant permissions to access resources. On the other hand, users and groups are security principals because they have a SID associated with them (that is, the **objectSID** attribute).

If an administrator needs to grant permissions to organizational unit members for a shared directory access (for instance), he must create a security group including the organizational unit members. This creates administrative work. To facilitate this, Microsoft adds an option to the user interface to populate a group based on the organizational unit members.

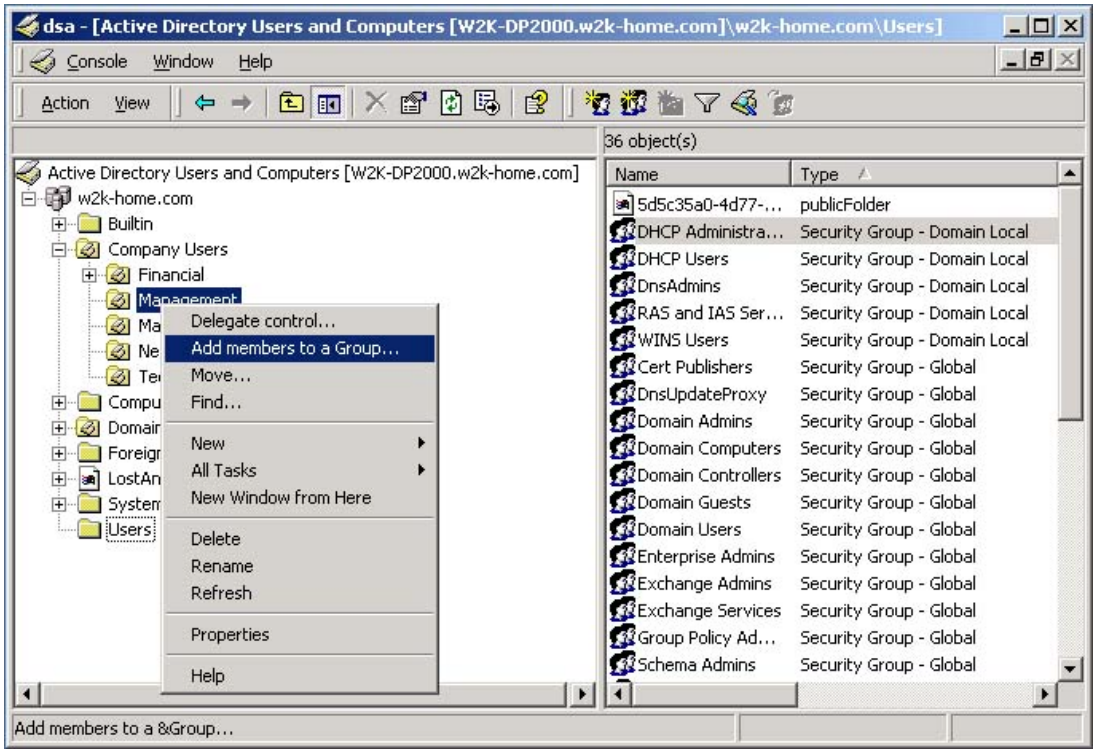


Figure 8 Adding organizational unit members to a group.

This GUI function facilitates the work but the administrator first needs to create the group planned to receive the members. Next, each time he adds new users, he must update each related group. Now, if the Active Directory has many organizational units, the administrator must repeat this operation for each of them. This can generate a non-negligible amount of administrative work.

It would be nice if a script could do this job. That is the purpose of Sample 18 on page 54. The script looks through the Active Directory for all existing organizational units and each time it finds one, it creates the associated group in that container. Then, the script adds all organizational unit members to the newly created group.

Moreover, the script will update new added users inside an organizational unit to the related group.

At this stage, organizational unit members are members of a group. So, an administrator is able to add that group to a resource access to grant permissions to all organizational unit members.

What happens if a resource used by organizational unit members is not available for a while? It would be nice to address organizational unit members via E-Mail to notify them of the problem. But an organizational unit is not mail-addressable; on the other hand, a group is mail-addressable. So, the next script will also enable the mail function to the created group to make it addressable from the Exchange 2000 Global Address List.

Creating a group object

Groups are distinct from organizational units. Organizational units are useful for creating an administrative hierarchy (delegation) or for setting group policies. Groups are used for granting access and creating distribution lists.

Groups and organizational units also differ in regard to the domain boundaries they apply to. Groups can be created to contain users, computers, a single domain, or multiple domains in a Forest. Organizational units represent a collection of objects (including group objects) only within the context of a single domain.

Under Windows 2000, several types of groups exist:

- **Universal**
  Members can come from any Windows NT/Windows 2000 domain in the Forest: Universal Groups, Global Groups and users from any domain in the Forest. Permissions can be granted in any domain in the Forest. Universal groups can be members of the following groups in the Forest: Local Groups and Universal Groups. If Universal groups are a security-enabled group, they are only usable when the domain is in native mode.

- **Global**
  Members can come from the domain containing the group: Global Groups and users (including contacts). Permissions can be granted in any domain in the Forest. Global groups can be members of any group in the forest: Global Groups, Local Groups, and Universal Groups.

- **Local**
  Members can come from any domain in the forest: Global Groups, Universal Groups and users from any domain in the forest and domain local groups from the domain containing the group. Permissions can be granted only on the domain containing the group. Local groups can only be a member of Local Groups in the domain containing the group.

When the administrator creates a group, he must determine if the group is security-enabled or not. If the group is not security-enabled, it is called a distribution group.

Some group types are available in regards of the Windows 2000 mode used (mixed or native mode), such as universal security-enabled groups.

Note: For more information on mixed and native mode, please refer to the White Paper by Micky Balladelli, "Active Directory - A technical overview".

To summarize the possible group operations with regards to the Windows 2000 mode:

In mixed mode:

- Global security groups can only contain user accounts.
- Domain local security groups can contain other global groups and user accounts.
- Universal security groups cannot be created.
- Universal distribution groups can be created.
- Only distribution groups can be nested.

In native mode:

- Universal groups are available as security groups and distribution groups.
- Full group nesting is allowed.
- Groups can be converted freely between security groups and distribution groups.
- Global groups and domain local groups can be converted to universal groups.

To create a group via ADSI, the script must use flag values specifying the type of group to be created:

```
ADS_GROUP_TYPE_GLOBAL_GROUP          = &H2
ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP = &H4
ADS_GROUP_TYPE_UNIVERSAL_GROUP       = &H8
ADS_GROUP_TYPE_SECURITY_ENABLED      = &H80000000
```

So, if the script uses the following combination:

```
ADS_GROUP_TYPE_SECURITY_ENABLED Or
ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP
```

It will create a security-enabled Domain Local Group. This group corresponds to the current Local Domain group known under Windows NT 4.0.

To create groups, it can be helpful to determine the Windows 2000 operation mode. To know if a Windows 2000 Domain is in mixed mode or in native mode, the **ntMixedDomain** attribute of the selected domain must be read. A zero value means the domain is in native mode. A value of one means the domain is in mixed mode.

Better than a long description, here is Sample 18.

Sample 18 Creating groups for organizational unit members

```
1:<!-- Windows Script File creating a security group per Organizational Unit with -->
2:<!-- each OU user object member included in the created group                    -->
3:<!--                                                                            -->
4:<!-- Version 1.00 - Alain Lissoir                                               -->
5:<!-- Compaq Computer Corporation - Professional Services - Belgium -            -->
6:<!--                                                                            -->
7:<!-- Any comments or questions:                 EMail:alain.lissoir@compaq.com -->
8:
```

```
 9:<Package>
10:<job>
11:        <script language="VBScript" src="Include.vbs" />
12:
13:        <script language="VBScript" src="ADSearchFunction.vbs" />
14:        <script language="VBScript" src="VerifyMSExchangeFunction.vbs" />
15:        <script language="VBScript" src="EnableE-MailFunction (ADSI).vbs" />
16:        <script language="VBScript" src="PromptParametersFunction.vbs" />
17:
18:        <script language="VBScript">
19:
20:        Option Explicit
21:
22:        ' Error code to test when object doesn't exist (Used for group presence checking)
23:        Const LDAP_NO_SUCH_OBJECT               = &h80072030
24:
25:        ' ------------------------------------------------------------------------------
..:
35:        strExchangeOrganization = "Home Business"
36:        strExchangeAdminGroup = "First Administrative Group"
37:        strCountry = "BE"
..:
49:        ' ------------------------------------------------------------------------------
50:        ' Check if Exchange 2000 is installed and if the given organization is created too.
51:        boolEnableEmail = VerifyExchangeOrg (strExchangeOrganization)
52:
53:        ' Search for all the OU
54:        Set objResultList = ADSearch ("DefaultNamingContext", _
55:                                      "(objectClass=organizationalUnit)", _
56:                                      "ADsPath", _
57:                                      "subTree", _
58:                                      False)
..:
62:        For Each objResult in objResultList
63:            intColumnPosition = InStr (objResult, ":")
64:            If intColumnPosition Then
65:               Call AddOUMember(objResultList.Item (objResult))
66:            End If
67:        Next
68:
69:        ' ------------------------------------------------------------------------------
70:        Private Sub AddOUMember (strOUPath)
..:
78:                Set objOU = GetObject (strOUPath)
79:                objOU.GetInfo
..:
83:                ' We look for the first user object
84:                For Each varMember In objOU
85:                    If varMember.Class = "user" Then
86:                       boolUsersExist = True
87:                       ' We found at least one user,the group can be created.
88:                       Exit For
89:                    End If
90:                Next
91:
92:                ' Get the name of the OU to build the name of the group.
93:                strGroup = objOU.ou
94:
95:                If (boolUsersExist) Then
96:                   ' We have at least one user, we bind the group
97:
98:                   ' Continue even if the group doesn't exists
99:                   On Error Resume Next
100:                  ' Clear any prior error to avoid confusion during next test
```

```
101:                    Err.Clear
102:
103:                    Set objGroup = objOU.GetObject("group", "CN=" & strGroup)
104:                    If Err.Number = LDAP_NO_SUCH_OBJECT Then
105:                       ' The object doesn't exist yet, we create it.
106:                       Wscript.Echo "  Creating group '" & strGroup & "'"
107:                       Set objGroup = objOU.Create("group", "CN=" & strGroup)
108:                       objGroup.Put "groupType", ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP Or _
109:                                                 ADS_GROUP_TYPE_SECURITY_ENABLED
110:                       objGroup.Put "sAMAccountName", Left(strGroup, 20)
111:                       objGroup.Put "name", strGroup
112:                       objGroup.Put "displayName", strGroup
113:                       objGroup.Put "description", strGroup & " OU Security Group"
114:                       objGroup.SetInfo
115:                    Else
116:                       ' The group exists, we will update it.
117:                       Wscript.Echo "  Updating group '" & strGroup & "'"
118:                    End If
119:
120:                    ' Enable all attributes needed to address the created
121:                    ' group from Exchange 2000
122:                    If boolEnableEmail Then
123:                       ' Show any error for the E-Mail enabling process
124:                       On Error Goto 0
125:                       ' We do not give any Exchange server or Storage group
126:                       ' because there is no mailbox.
127:                       Call EnableEmailAddress (objGroup, _
128:                                                strExchangeOrganization, _
129:                                                strExchangeAdminGroup, _
130:                                                "", _
131:                                                "", _
132:                                                NO_CREATE_MB)
133:                    End If
134:
135:                    ' Suppress any error if user is already in the group
136:                    On Error Resume Next
137:                    ' All users present in the OU are added to the created/retrieved group.
138:                    For Each varMember In objOU
139:                        If varMember.Class = "user" Then
140:                           WScript.Echo "   Adding '" & varMember.Name & _
141:                                        " (" & varMember.displayName & _
142:                                        ")' to associated security group."
143:                           objGroup.Add (varMember.ADsPath)
144:                        End If
145:                    Next
146:
147:                    ' Commit the membership assignment
148:                    objGroup.SetInfo
149:
150:                    WScript.DisconnectObject objGroup
151:                    Set objGroup = Nothing
152:                 Else
153:                    ' No user in the OU, so we delete the associated group.
154:
155:                    ' Avoid any problem if the Group doesn't exists
156:                    On Error Resume Next
157:                    ' Clear any prior error to avoid confusion during next test
158:                    Err.Clear
159:
160:                    Set objGroup = objOU.GetObject("group", "CN=" & strGroup)
161:                    If Err.Number = LDAP_NO_SUCH_OBJECT Then
162:                       ' Do nothing, the group doesn't exist because no user exist.
163:                       Wscript.Echo "  No user found."
164:                    Else
```

```
165:                        ' We delete the existing group because no user were found in the OU.
166:                        Wscript.Echo "  Deleting group '" & strGroup & "'"
167:                        objOU.Delete "group", "CN=" & strGroup
168:                    End If
169:
170:            End If
171:
172:            Set objOU = Nothing
173:
174:        End Sub
175:
176:        </script>
177:</job>
178:
179:</Package>
```

In the beginning of the script (line 11 and 16), there are five separate functions included to help in this task.

- **Include.vbs:** This function defines all labels and constants (such as Group constant definitions) used in other functions.
- **ADSearchFunction.vbs:** This function helps to locate any **organizationalUnit** object class present in the Active Directory.
- **VerifyMSExchangeFunction.vbs:** This function verifies the presence of Exchange 2000 in the forest and checks if the given Exchange organization is present.
- **EnableE-MailFunction (ADSI).vbs:** This function enables the E-Mail function on an object. Moreover, this function will help to create an Exchange 2000 Mailbox for Sample 22 on page 68**.**
- **PromptParametersFunction.vbs:** This function prompts the user for specific parameters.

We will come back to these functions later.

One of the purposes is to enable E-Mail addressing on the created groups. Therefore, the script defines the Exchange Organization and the Exchange 2000 Administrative Group (lines 35 and 36). After prompting the user for some external parameters (line 43 to 50 not visible here), it checks the presence of Exchange 2000 and the existence of the given Exchange Organization (line 51).

Next, the script reuses the well-known 'ADSearch' function to locate all the organizational units in the current domain (lines 54 to 58). Then, it enumerates the Dictionary object to extract each organizational unit found (lines 62 to 67). For each organizational unit, it calls the 'AddOUMember' function (line 65).

The 'AddOUMember' function is the intelligence of the script. It uses the organizational unit **ADsPath** as a parameter. The first operation is to bind to the organizational unit (line 78) to see if some users exist in the container (lines 84 to 90). If there is no user in the organizational unit, it is useless to create a group with no members. Note that if no user exists in the organizational unit, the script tries to delete the existing group because there is no reason anymore to maintain the group in the Active Directory (line 161 to 168).

Next, the script binds to the existing group in the organizational unit (line 103). The group has the same name as the organizational unit (design choice). If the group exists, the script updates the group. If the group does not exist, it creates the group (lines 105 to 114). Note the flag value combination to determine the group type created (lines 108 and 109). During the group creation process, the mandatory attributes and some optional attributes are created.

After, the creation process (if the group does not exist) or the bind process (if the group already exists), the actions taken are the same for an existing group or a new group. The mail function of the group is enabled if the Exchange 2000 tests were successful (lines 122 to 133). Next, the script adds each user present in the organizational unit to the group (lines 138 to 145). Then, changes are committed to the Active Directory (line 148).

Working with Exchange 2000

Checking the presence of Exchange 2000, checking the presence of an Exchange Organization name

When performing an Exchange 2000 installation, significant changes are made to the Active Directory. The schema is modified; new objects and new attributes are created. Checking the presence of some specific objects related to Exchange 2000 is the way to test if Exchange 2000 is present in the Forest. These objects are located in the **configurationNamingContext**.

This context contains a container called 'Services'. This container contains all the services available in the Forest. If Exchange 2000 is installed, the script will find a container called 'Microsoft Exchange'.

In this last container, the created Exchange 2000 Organization is present. The script can determine the organization object by looking for an **objectClass** called **msExchOrganizationContainer**. Based on this structure, it is easy to check the presence of an Exchange Organization just by looking for the desired **objectClass** and name.

This is the purpose of Sample 19 below.

Sample 19 Checking the presence of Exchange 2000 with a specific Exchange Organization name.

```
 1:' VB Script to verify the presence of Exchange 2000 and a given organization    '
 2:'                                                                               '
 3:' Version 1.00 - Alain Lissoir                                                  '
 4:' Compaq Computer Corporation - Professional Services - Belgium -               '
 5:'                                                                               '
 6:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 7:
 8:Option Explicit
 9:
10:' ---------------------------------------------------------------------------------------
11:Private Function VerifyExchangeOrg (strExchangeOrganization)
..:
18:        Set ObjRoot = GetObject("LDAP://RootDSE")
19:        strConfigNC = ObjRoot.Get("configurationNamingContext")
..:
23:        On Error Resume Next
24:        Set objExchangeContainer = GetObject ("LDAP://CN=Microsoft Exchange,CN=Services," & _
25:                                              strConfigNC)
26:        If Err.Number Then
27:           ' Exchange container in services not present, Exchange 2000 is not installed
28:           VerifyExchangeOrg = False
29:           Exit Function
30:        End If
31:
32:        For Each objExchangeChild In objExchangeContainer
33:            If (objExchangeChild.Class = "msExchOrganizationContainer") And _
34:               (objExchangeChild.Get ("cn") = strExchangeOrganization) Then
35:                  Wscript.Echo "Found Exchange Organization called '" & _
36:                               objExchangeChild.Get ("cn") & "'."
37:                  Wscript.Echo
38:                  VerifyExchangeOrg = True
39:                  Exit Function
40:            End If
41:        Next
42:
43:        VerifyExchangeOrg = False
44:
```

```
45:End Function
```

Enabling E-Mail on objects

Enabling E-Mail on an object requires defining more attributes associated with the object. To determine the value assigned to those new attributes, more information about the Exchange 2000 installation needs to be taken from the Active Directory. Once this information is known, it is possible to construct, for instance, the SMTP address, the X400 address, the mail alias and so on. This is the purpose of Sample 20 below.

Sample 20 Enabling E-Mail on object for Exchange 2000 with ADSI

```
 1:' VB Script enabling Email functions on Groups or Users objects.    '
 2:' Email can be enabled with or without an Exchange 2000 mailbox     '
 3:' for a user object only.                                           '
 4:'                                                                   '
 5:' The Exchange 2000 mailbox creation via ADSI is a method not       '
 6:' supported by Microsoft.                                           '
 7:' CDO for Exchange Management (CDOEXM) must be used instead.        '
 8:'                                                                   '
 9:' Version 1.01 - Alain Lissoir                                      '
10:' Compaq Computer Corporation - Professional Services - Belgium -   '
11:'                                                                   '
12:' Any comments or questions:         EMail:alain.lissoir@compaq.com '
13:
14:Option Explicit
15:
16:Const NO_CREATE_MB = 0
17:Const CREATE_MB = 1
18:
19:' ------------------------------------------------------------------------------------------
20:Private Sub EnableEmailAddress (objObject, _
21:                                strExchangeComputer, _
22:                                strOrganization, _
23:                                strExchangeAdminGroup, _
24:                                strExchangeStorageGroup, _
25:                                strExchangeMailboxStore, _
26:                                boolMB)
..:
45:        Set ObjRoot = GetObject("LDAP://RootDSE")
46:        strRootDomainNC = objRoot.Get("RootDomainNamingContext")
47:        strConfigNC = ObjRoot.Get("configurationNamingContext")
..:
51:        ' ------------------------------------------------------------------------------
52:        Set objRootDomainNC = GetObject("LDAP://" & strRootDomainNC)
53:
54:        ' Retrieve a constructed property, so 1st we do a GetInfoEx
55:        objRootDomainNC.GetInfoEx Array("canonicalName"), 0
56:        strRootDNSDomainName = objRootDomainNC.Get("canonicalName")
57:        ' Remove the / at the end
58:        strRootDNSDomainName = Mid (strRootDNSDomainName, 1, Len(strRootDNSDomainName) - 1)
..:
63:        ' Build the path with the Microsoft Exchange organization path
64:        strOrganizationDN = "CN=" & strOrganization & _
65:                            ",CN=Microsoft Exchange,CN=Services," & strConfigNC
66:
67:        ' ------------------------------------------------------------------------------
68:        Select Case objObject.Class
69:               Case "user"
70:                    strRecipient = EliminateSpaces (LCase (objObject.FirstName) & _
71:                                                    "." & LCase (objObject.LastName))
72:                    ' Set the Alias name
```

```
 73:                     objObject.put "mailNickName", strRecipient
 74:
 75:                     ' Build the SMTP Address
 76:                     strSMTPAddress = strRecipient & "@" & strRootDNSDomainName
 77:                     ' Build the X400 Address
 78:                     strX400Address = "c=" & strCountry & _
 79:                                      ";a= " & _
 80:                                      ";p=" & Left (strOrganization, 16) & _
 81:                                      ";o=Exchange" & _
 82:                                      ";s=" & Lcase (objObject.LastName) & _
 83:                                      ";g=" & Lcase (objObject.FirstName) & _
 84:                                      ";"
 85:
 86:                     ' Set in which GAL to show the user
 87:                     strDefaultGAL = "CN=Default Global Address List," & _
 88:                                 "CN=All Global Address Lists,CN=Address Lists Container," & _
 89:                                 strOrganizationDN
 90:                     objObject.PutEx ADS_PROPERTY_UPDATE, "showInAddressBook", _
 91:                                     Array(strDefaultGAL)
 92:
 93:                     ' Set the legacy Exchange distinguished name of the created user
 94:                     strLegacyExchangeDN = "/o=" & strOrganization & _
 95:                                           "/ou=" & strExchangeAdminGroup & _
 96:                                           "/cn=Recipients/cn=" & strRecipient
 97:
 98:                Case "group"
 99:                     strRecipient = EliminateSpaces (objObject.Get ("sAMAccountName"))
100:
101:                     ' Set the Alias name
102:                     objObject.put "mailNickName", strRecipient
103:
104:                     ' Build the SMTP Address
105:                     strSMTPAddress = strRecipient & "@" & strRootDNSDomainName
106:                     ' Build the X400 Address
107:                     strX400Address = "c=" & strCountry & _
108:                                      ";a= " & _
109:                                      ";p=" & Left (strOrganization, 16) & _
110:                                      ";o=Exchange" & _
111:                                      ";s=" & strRecipient & _
112:                                      ";"
113:
114:                     ' Set in which GAL to show the group
115:                     strDefaultGAL = "CN=Default Global Address List," & _
116:                                 "CN=All Global Address Lists,CN=Address Lists Container," & _
117:                                 strOrganizationDN
118:                     strGroupGAL = "CN=All Groups,CN=All Address Lists," & _
119:                                   "CN=Address Lists Container," & _
120:                                   strOrganizationDN
121:                     objObject.PutEx ADS_PROPERTY_UPDATE, "showInAddressBook", _
122:                                     Array(strGroupGAL, strDefaultGAL)
123:
124:                     ' Set the legacy Exchange distinguished name of the created group
125:                     strLegacyExchangeDN = "/o=" & strOrganization & _
126:                                           "/ou=" & strExchangeAdminGroup & _
127:                                           "/cn=Recipients/cn=" & strRecipient
...:
131:        End Select
132:
133:        objObject.PutEx ADS_PROPERTY_UPDATE, "proxyAddresses", _
134:                        Array("X400:" & strX400Address, "SMTP:" & strSMTPAddress)
135:
136:        ' Other LDAP value with X400 address
137:        objObject.put "textEncodedORAddress", strX400Address
138:
```

```
139:         ' Other LDAP value with SMTP Address
140:         objObject.put "mail", strSMTPAddress
141:
142:         objObject.put "msExchHideFromAddressLists", False
143:
144:         objObject.put "legacyExchangeDN", strLegacyExchangeDN
145:
146:         objObject.SetInfo
147:
148:         ' ---------------------------------------------------------------------------
149:         If (objObject.Class = "user" And boolMB) Then
150:            objObject.put "mDBUseDefaults", True
151:
152:            strHomeMTA = "CN=Microsoft MTA,CN=" & strExchangeComputer & _
153:                         ",CN=Servers,CN=" & strExchangeAdminGroup & _
154:                         ",CN=Administrative Groups," & strOrganizationDN
155:            objObject.put "HomeMTA", strHomeMTA
156:
157:            strHomeMDB = "CN=" & strExchangeMailboxStore & ",CN=" & _
158:                         strExchangeStorageGroup & _
159:                         ",CN=InformationStore,CN=" & strExchangeComputer & _
160:                         ",CN=Servers,CN=" & strExchangeAdminGroup & _
161:                         ",CN=Administrative Groups," & strOrganizationDN
162:            objObject.put "HomeMDB", strHomeMDB
163:
164:            ' Set the Exchange Home server of the created user
165:            strExchangeComputerDN = "/o=" & strOrganization & _
166:                                    "/ou=" & strExchangeAdminGroup & _
167:                                    "/cn=Configuration/cn=Servers/cn=" & _
168:                                    strExchangeComputer
169:            objObject.put "msExchHomeServerName", strExchangeComputerDN
170:            objObject.SetInfo
171:
172:            Wscript.Echo "  Successfully created Microsoft " & _
173:                         "Exchange 2000 Mailbox for user '" & _
174:                         objObject.Get ("cn") & "'."
175:
176:            ' ---------------------------------------------------------------------------
177:            ' Create the Exchange 2000 Security Descriptor
178:            SetSD objObject, _
179:                  Array ("Self", _
180:                                 ADS_ACETYPE_ACCESS_ALLOWED, _
181:                                 RIGHT_MB_READ_PERMISSIONS Or _
182:                                  RIGHT_MB_FULL_MB_ACCESS Or _
183:                                  RIGHT_MB_SEND_AS, _
184:                                 CONTAINER_INHERIT_ACE), _
185:                  MB_OBJECT
186:         Else
187:            ' If mail-enabled object, initialize targetAddress with the SMTP Address
188:            objObject.put "targetAddress", "SMTP:" & strSMTPAddress
189:
190:            ' Make this for the X400 address if necessary (instead of SMTP:)
191:            ' objObject.put "targetAddress", "X400:" & strX400Address
192:            objObject.SetInfo
193:
194:            Wscript.Echo "  Successfully enabled Microsoft Exchange 2000 E-Mail for '" & _
195:                         objObject.Get ("cn") & "'."
196:
197:         End If
198:
199:End Sub
...:
...:
...:
```

The script starts by getting some information from the Active Directory, such as the **RootDomainNamingContext** (line 46). The **RootDomainNamingContext** makes it possible to determine the **canonicalName** (lines 55 and 56). This name is used to construct the domain name for the SMTP Address construction (lines 76 and 105).

The **configurationNamingContext** is also important for building the Exchange organization distinguished name. With the help of the Exchange Organization name, the Exchange Organization distinguished name is created (lines 64 and 65). The Exchange Organization name is also used to build the private management domain name of the X400 address (lines 78 and 107). The Exchange organization distinguished name is used for the miscellaneous attribute initialization:

- **DefaultGAL** (lines 87 and 115)
- **GroupGAL** (line 118) for group objects only
- **HomeMTA** (line 152)
- **HomeMDB** (line 157)

It is important to have an alias for the enabled mail objects. The alias is constructed from the **sAMAccountName** attribute for group objects (line 99). For the user objects, the alias is constructed from the first name and last name (line 70). The script eliminates all spaces to construct this attribute. Next, the **mailNickName** attribute is assigned (line 73 for user objects or line 102 for group objects). The **mailNickName** attribute is the Exchange alias.

The object class determines how to set addresses for a given object. For instance, for a group, there is no first name. This requires an adapted method to construct the addresses. This is why the object type is tested on its class (lines 69 and 98).

- **For a user class:**

  The strRecipient variable is initiated at line 70. This variable contains the "FirstName.LastName" string. The script will use that variable content as the SMTP recipient name. The domain name comes from the **canonicalName** attribute (lines 55 and 56 above).

  The X400 address is constructed from different attributes:

  - **The country:** The strCountry variable (line 78) is a public variable defined in the script calling the 'EnableE-MailFunction (ADSI).vbs' function (see Sample 22 on page 68).

Note: It was decided to put all user parameters in public variables with an explicit name. Using public variables inside a function is not the cleanest style of programming. Ideally, it was possible to use an array containing all the user attributes. This is a better programming method. Unfortunately, in that case, the meaning and the origin of the values are not so clear for the reader. The underlying philosophy of this document is to ease the understanding. It was decided to use an explicit variable for each user attribute. This makes the script easier to read.

  - **The private management domain:** As explained before, this value gets the Exchange Organization name passed to the script function as a parameter (line 80). Note that the private management domain is truncated on 16 positions.
  - **The organization:** The 'Exchange' keyword is assigned to this X400 field (line 81).
  - **The surname and the givenname:** They are assigned from two public variables (strLastName and strFirstName). Both variables must be defined in the script calling the 'EnableE-MailFunction (ADSI).vbs' function (Lines 82 and 83).

  Both addresses (SMTP and X400) are assigned to the multi-valued **proxyAddresses** attribute (lines 133 and 134). Next, the script assigns, respectively, the X400 address and the SMTP address to the **textEncodedORAddress** attribute and the **mail** attribute (lines 137 and 140).

The Exchange Organization distinguished name is used to complete the distinguished name of the default Global Address List (line 89). This default Global Address List is set to the **showInAddressBook** attribute (line 90), a multi-valued attribute able to receive more than one Address book distinguished name value.

Finally, it is necessary to construct an attribute called **legacyExchangeDN.** This attribute contains a legacy Exchange 5.5 distinguished name with **'o=OrgName'** property (for Organization), **'ou=SiteName'** property (for Site), '**cn=Recipients**' property (for the recipients container) and '**cn=Alias'**. For the user object, the mail alias used is the **mailNickName** attribute (lines 94 to 96) contained in the strRecipient variable. The attribute assignment in the Active Directory is completed at line 144.

- **For a group class:**

  Like the **user** class, the **group** class also has an SMTP address. The difference resides only in the recipient name construction for the **group** class. Because there is no first name and last name, the script builds an alias from the **sAMAccountName** (line 99). The script uses this alias to define the recipient name of the SMTP address (line 105). The domain name comes from the **canonicalName** attribute (lines 55 and 56, see above) like the **user** class.

  For the X400 address, there is the same problem as the SMTP address. There is no first name for a **group** class, so the script uses the alias again. The alias is assigned to the **surname** field of the X400 address (line 111).

  Once the SMTP address and X400 address are constructed, they are saved in the multi-valued attribute **proxyAddresses** (line 137 and 140). The **mail** (SMTP) and **textEncodedORAddress** (X400) are also assigned with their respective address types.

  In the same way as for the user class, the script builds the distinguished names for the address lists with the Exchange Organization distinguished name. For the **group** class, two address lists are defined: the default address list known as the Global Address List (line 115) and the Group address list (line 118), which is a Global Address List view on Windows 2000 E-Mail enabled groups (distribution lists). Next, the **showInAddressBook** attribute is saved with these address list distinguished names (line 121).

  Finally, to construct the **legacyExchangeDN** attribute for the group object, the alias built on line 99 is used again (lines 125 to 127). The final attribute assignment is completed at line 144.

On line 142, the **msExchHideFromAddressLists** is set to False in order to show this address in the address list. This is the GUI property equivalent to 'hide from address book' in Exchange 5.5.

At line 149, the function makes a test to determine if the object class is a user and if the mail operation to perform is related to the creation of a mail-enabled object or mailbox-enabled object. If the variable boolMB is True and the object class is a user, the object will be mailbox-enabled (lines 150 to 185). In any other case the object will be a mail-enabled object (lines 187 to 195).

To create a mail-enabled object, the **targetAddress** attribute must be set (line 188). This attribute can contain the SMTP address (line 188) or the X400 address (line 191) of the recipient. This is a single-valued attribute.

At this stage of the script, the object is E-Mail enabled. This means that the object is addressable from the Exchange 2000 address book. This does not mean that the object has an associated mailbox. To create an Exchange 2000 mailbox, some more attributes need to be set.

Creating the Exchange 2000 mailbox with ADSI

Under Exchange 5.5, creating a mailbox is equal to creating an object under the Recipient container with some specific attribute settings. Under Windows 2000, this is different because there is no separate directory for the mailboxes. This means that it is not necessary to create a new object in order to create a mailbox. The principle is to set some user object attributes that define the mailbox settings. These attributes are the mailbox location (**HomeMTA**), the mailbox database location (**HomeMDB**), the server hosting the mailbox (**msExchHomeServerName**) and the mailbox access rights (**msExchMailboxSecurityDescriptor**).

To come back to the Sample 20 on page 59, lines 149 to 186 define the miscellaneous attributes to enable the mailbox definition in the directory. The **mDBUseDefaults** attribute has a flag referring the mailbox default settings to the Information Store defaults (line 150).

The **HomeMTA** attribute must also be set with the Home MTA distinguished name. This value is constructed from several elements such as the Exchange computer name, the Exchange Admin group and the Exchange organization distinguished name (lines 152 to 155).

The **HomeMDB** attribute determines the Information Store location hosting the mailbox. This value is constructed from several elements such as the Exchange Storage group, the Exchange computer, the Exchange Admin group and the Exchange organization distinguished name (lines 157 to 162).

The **msExchHomeServerName** attribute is constructed from the same set of information as the **HomeMTA** and the **HomeMDB** attributes. This value is the Exchange computer distinguished name hosting the mailbox (lines 165 to 169). Warning, this attribute is an Exchange computer distinguished name and must not be confused with the DC distinguished name also located in the Configuration context but below the site level definition.

The **msExchMailboxSecurityDescriptor** contains the security descriptor defining the mailbox access rights. The script code will complete the mailbox creation process by assigning the necessary permissions (lines 178 to 185) with the SetSDFunction.vbs function (See Sample 17 “Creating a security descriptor” on page 49 above). This is why the function is included at line 22 of Sample 22 on page 68.

The Sample 20 stores all the ACE entries to set the access rights on the mailbox (lines 179 to 185) and calls the SetSD function (line 178). See Table 6 on page 93 for Exchange 2000 flag values. The security descriptor set in the script is the same as the security descriptor set by the Administrator GUI by default. Once this operation is complete, the Exchange 2000 mailbox definition is created in the Active Directory.

The routine is the complete set of operations to create an Exchange 2000 mailbox-enabled user with ADSI. This routine will be required in a following script sample (Sample 22 on page 68).

Creating the Exchange 2000 mailbox with CDO for Exchange Management

With Exchange 2000, Microsoft has included a new version of the Collaboration Data Object (CDO) to help in the management of Exchange. This is formally called CDO for Exchange Management (CDOEXM). CDOEXM provides objects and interfaces for the management of many Exchange 2000 components. One of these components is the mailbox. CDOEXM acts as an extension for ADSI to ease the creation of and access to mailbox definitions stored in the Active Directory.

CDOEXM is an important companion to ADSI when working with Exchange 2000.

Note: This document does not focus on Exchange 2000 and its related COM components. For more information about CDO for Exchange management, the reader may refer to other publications (from the same author) available on the Compaq Active Answer web site.

http://vcmproapp02.compaq.com/ActiveAnswers/Global/en/solutions.1128/offline.8588/default.asp

- "Part 1 - Introduction to the use of Exchange 2000 with Windows Script Host"
- "Part 2 - Managing Exchange with Scripts, Advanced Topics"

The Sample 21 below is using CDOEXM for the Exchange 2000 mailbox creation. This sample is given for reference only. The script is using the exact same parameters as Sample 20 on page 59. The purpose is to ease the switch of the EnableEmailAddress function from the ADSI method (Sample 20) to the CDOEXM method (Sample 21).

Note: It is important to know that the only supported method by Microsoft is the CDOEXM method. Even if the ADSI method works fine with Exchange 2000, there is no guarantee that this method will continue to work with the future versions of Exchange. CDOEXM encapsulates the business logic required to create a fully functional mailbox. The ADSI method uses a raw access method to create the mailbox and does not protect the code from any change made in the future Exchange versions.

Sample 21 Enabling E-Mail on object for Exchange 2000 with CDOEXM

```
 1:' VB Script enabling Email functions on Groups or Users objects.    '
 2:' Email can be enabled with or without an Exchange 2000 mailbox     '
 3:' for a user object only.                                           '
 4:'                                                                   '
 5:' Version 1.00 - Alain Lissoir                                      '
 6:' Compaq Computer Corporation - Professional Services - Belgium -   '
 7:'                                                                   '
 8:' Any comments or questions:         EMail:alain.lissoir@compaq.com '
 9:
10:Option Explicit
11:
12:Const NO_CREATE_MB = 0
13:Const CREATE_MB = 1
14:
15:Const OBJECT_ALREADY_MAIL_ENABLED = &h80004005
16:
17:' ------------------------------------------------------------------------------------------
18:Private Sub EnableEmailAddress (objObject, _
19:                                strExchangeComputer, _
20:                                strOrganization, _
21:                                strExchangeAdminGroup, _
22:                                strExchangeStorageGroup, _
23:                                strExchangeMailboxStore, _
24:                                boolMB)
..:
35:        Set ObjRoot = GetObject("LDAP://RootDSE")
36:        strRootDomainNC = objRoot.Get("RootDomainNamingContext")
37:        strConfigNC = ObjRoot.Get("configurationNamingContext")
38:        WScript.DisconnectObject ObjRoot
39:        Set ObjRoot = Nothing
40:
41:        ' ------------------------------------------------------------------------------
42:        Set objRootDomainNC = GetObject("LDAP://" & strRootDomainNC)
43:
44:        ' Retrieve a constructed property, so 1st we do a GetInfoEx
45:        objRootDomainNC.GetInfoEx Array("canonicalName"), 0
```

```
46:         strRootDNSDomainName = objRootDomainNC.Get("canonicalName")
47:         ' Remove the / at the end
48:         strRootDNSDomainName = Mid (strRootDNSDomainName, 1, Len(strRootDNSDomainName) - 1)
49:
50:         WScript.DisconnectObject objRootDomainNC
51:         Set objRootDomainNC = Nothing
52:
53:        ' Just make this to avoid a CDOEXM error if already mail-enabled or mailbox-enabled.
54:         On Error Resume Next
55:
56:         ' ----------------------------------------------------------------------------
57:         Select Case objObject.Class
58:                Case "user"
59:                     If boolMB Then
60:                        objObject.CreateMailbox "LDAP://" & strAccountComputer & "/" & _
61:                                                "cn=" & strExchangeMailboxStore  & "," & _
62:                                                "cn=" & strExchangeStorageGroup & "," & _
63:                                                "cn=InformationStore," & _
64:                                                "cn=" & strExchangeComputer & ",cn=Servers," & _
65:                                                "cn=" & strExchangeAdminGroup & "," & _
66:                                                "cn=Administrative Groups," & _
67:                                                "cn=" & strOrganization & "," & _
68:                                                "cn=Microsoft Exchange,CN=Services," & strConfigNC
69:                        If Err.Number = OBJECT_ALREADY_MAIL_ENABLED Then
70:                           WScript.Echo Err.Description
71:                           Err.Clear
72:                        End If
73:                     Else
74:                        strRecipient = EliminateSpaces (LCase (objObject.FirstName) & _
75:                                                        "." & LCase (objObject.LastName))
76:
77:                        strSMTPAddress = strRecipient & "@" & strRootDNSDomainName
78:                        strX400Address = "c=" & strCountry & _
79:                                         ";a= " & _
80:                                         ";p=" & Left (strOrganization, 16) & _
81:                                         ";o=Exchange" & _
82:                                         ";s=" & LCase (objObject.LastName) & _
83:                                         ";g=" & LCase (objObject.FirstName) & _
84:                                         ";"
85:
86:                        ' If mail-enabled object, initialize targetAddress with the SMTP Address
87:                        objObject.MailEnable "SMTP:" & strSMTPAddress
88:                        If Err.Number = OBJECT_ALREADY_MAIL_ENABLED Then
89:                           WScript.Echo Err.Description
90:                           Err.Clear
91:                        End If
92:
93:                        ' Make this for the X400 address if necessary (instead of SMTP:)
94:                        ' objObject.MailEnable "X400:" & strX400Address
95:                        ' If Err.Number = OBJECT_ALREADY_MAIL_ENABLED Then
96:                        '    WScript.Echo Err.Description
97:                        '    Err.Clear
98:                        ' End If
99:                     End If
100:
101:               Case "group"
102:                    ' If mail-enabled object, initialize targetAddress with the SMTP Address
103:                    objObject.MailEnable
104:                    If Err.Number = OBJECT_ALREADY_MAIL_ENABLED Then
105:                       WScript.Echo Err.Description
106:                       Err.Clear
107:                    End If
108:
109:               Case Else
```

```
110:                    Exit Sub
111:        End Select
112:
113:        ' Reset the 'On Error Resume Next' statement.
114:        On Error Goto 0
115:
116:        objObject.SetInfo
117:
118:        If objObject.Class = "user" And boolMB Then
119:           Wscript.Echo "  Successfully created Microsoft " & _
120:                        "Exchange 2000 Mailbox for user '" & _
121:                        objObject.Get ("cn") & "'."
122:        Else
123:           Wscript.Echo "  Successfully enabled Microsoft Exchange 2000 E-Mail for '" & _
124:                        objObject.Get ("cn") & "'."
125:        End If
126:End Sub
...:
...:
...:
```

Building Active Directory user accounts and Exchange 2000 mailboxes

This last script is the final use of all the scripts written before. The creation of the Active Directory user object and the Exchange 2000 associated mailbox completely re-use the prior developed functions. This is where the script will take the benefit of all the "general" functions written before.

The core logic of the script

The only new task is the user object creation process (with its associated attributes). This involves the following operations:

- Creation of miscellaneous names such as first name, last name, SAM name and full name
- User profile and Home directory definitions
- User profile and home directory creation on the File System
- Share creation for the home directory
- Default Security Settings such as:
  - Logon and password policies
  - User rights on the Home Directory
  - User rights on the Profile Directory

With all previously developed functions, the script will handle the following tasks:

- Setting access rights on the Home directory
- Setting access rights on the Profile directory
- Checking the presence of Exchange 2000 in the Configuration context
- Checking the presence of Exchange 2000 Organization name given
- Creating the Exchange 2000 mailbox-enabled or mail-enabled attribute definition.

All data needed for the user object creation and its associated attributes will come from an Excel sheet. Sample 22 below handles these operations.

Sample 22 Creating users under Windows 2000 with their associated Exchange 2000 mailboxes

```
 1:<!-- Windows Script File to create users in Windows 2000 Active Directory         -->
 2:<!-- (creating users, Home and Profile Directories, Home share and Exchange 2000  -->
 3:<!-- mailboxes with security access)                                              -->
 4:<!--                                                                             -->
 5:<!-- Version 1.00 - Alain Lissoir                                                -->
 6:<!-- Compaq Computer Corporation - Professional Services - Belgium -             -->
 7:<!--                                                                             -->
 8:<!-- Any comments or questions:                   EMail:alain.lissoir@compaq.com -->
 9:
10:<job>
11:        <script language="VBScript" src="..\Functions\Include.vbs" />
12:
13:        <script language="VBScript" src="..\Functions\PromptParametersFunction.vbs" />
14:
15:        <script language="VBScript" src="..\Functions\VerifyMSExchangeFunction.vbs" />
16:        <script language="VBScript" src="..\Functions\EnableE-MailFunction (ADSI).vbs" />
17:
18:        <script language="VBScript" src="..\Functions\AddRightFunction.vbs" />
19:        <script language="VBScript" src="..\Functions\RemoveRightFunction.vbs" />
20:
21:        ' Function needed only if using "EnableE-MailFunction (ADSI).vbs"
22:        <script language="VBScript" src="..\Functions\SetSDFunction.vbs" />
```

```
  23:
  24:         <script language="VBScript" src="..\Functions\CreateFolderFunction.vbs" />
  25:         <script language="VBScript" src="..\Functions\CreateShareFunction.vbs" />
  26:
  27:         <script language="VBScript">
  28:
  29:         Option Explicit
  30:
  31:         ' ------------------------------------------------------------------------------
  32:         ' Constants Declaration Section
  33:
  34:         ' ------------------------------------------------------------------------------
  35:         ' Account settings for user creation
  36:         ' Error code to test when object user doesn't exist
  37:         Const LDAP_NO_SUCH_OBJECT = &h80072030
  38:
  39:         ' Put variables installation parameters in constant labels
  40:         ' (Easier for future adaptation)
  41:
  42:         ' Machine member of the Domain homing the user accounts
  43:         Const cAccountComputer          = "MYW2KDC"
  44:         ' Machine running Microsoft Exchange 2000
  45:         Const cExchangeComputer         = "MYW2KDC"
  46:         ' Organization parameters with specific Exchange 2000 Admin Group and Storage
  47:         Const cExchangeOrganization     = "First Organization"
  48:         Const cExchangeAdminGroup       = "First Administrative Group"
  49:         Const cExchangeStorageGroup     = "First Storage Group"
  50:         Const cExchangeMailboxStore     = "Mailbox Store (A)"
  51:
  52:         ' Misc. user settings for default assignements
  53:         Const cLogonScript              = "LOGON_SCRIPT.CMD"
  54:         Const cDefaultPassword          = "password"
  55:         Const cHomeDrive                = "U:"
  56:
  57:         ' Misc. settings for Profile and Home Directory paths (Local and Remote via UNC)
  58:         ' Letter mapping used locally on the server having the homedirectories.
  59:         ' This is the local path base used on the server himself.
  60:         ' (From a local filesystem point of view)
  61:         Const cHomeDirLocalAccess       = "C:\DATA\HOME\"
  62:
  63:         Const cProfileShare             = "\Profile$\"
  64:         Const cHomeShare                = "\HOME$\"
  65:
  66:         ' Default Excel datasheet to use when nothing is given on the command line
  67:         Const cXLDataSheetName          = "C:\Data\Excel\PopulateUsers.xls"
 ...:
 140:
 141:         ' Path for a remote access to the filesystem having homedirectories
 142:         ' This is the path to use to access the base directory containing
 143:         ' the Home directory. You should UNC because WSH script will probably
 144:         ' not run on the machine homing directories.
 145:         ' This supposes that the base directory is shared.
 146:         '
 147:         ' This value is not prompted
 148:         strHomeDirRemoteAccess = "\\" & strAccountComputer & cHomeShare
 149:         ' This value is not prompted
 150:         strProfileDirRemoteAccess = "\\" & strAccountComputer & cProfileShare
 151:         ' This value is not prompted
 152:         strHomeDirLocalAccess = cHomeDirLocalAccess
 ...:
 ...:         < SKIP THE MISC. PARAMETERS PROMPTING >
 ...:
 197:         ' ------------------------------------------------------------------------------
 198:         ' Check if Exchange 2000 is installed and if the given organization is created too.
```

```
199:        boolEnableEmail = VerifyExchangeOrg (strExchangeOrganization)
200:
201:        ' ------------------------------------------------------------------------------
202:        ' Get the Default Domain name where we will create Windows 2000 users
203:        ' based on the computer name given
204:        Set objRoot = GetObject("LDAP://" & strAccountComputer & "/RootDSE")
205:        strAccountDomain = objRoot.Get("DefaultNamingContext")
206:        strRootDomain = objRoot.Get("RootDomainNamingContext")
207:        WScript.DisconnectObject objRoot
208:        Set objRoot = Nothing
209:
210:        ' Bind to the current account Domain to get its Netbios name
211:        Set objDomain = GetObject ("LDAP://" & strAccountDomain)
212:
213:        ' The domain name will be used to set access rights on directories
214:        strAccountNBDomain = UCase (objDomain.get ("name"))
215:        WScript.DisconnectObject objDomain
216:        Set objDomain = Nothing
217:
218:        ' Bind to the Root Domain to get its canonical name for UPN contruction
219:        Set objRootDomain = GetObject("LDAP://" & strRootDomain)
220:
221:        ' Retrieve a constructed property, so 1st we do a GetInfoEx (For UPN construction)
222:        objRootDomain.GetInfoEx Array("canonicalName"), 0
223:        strRootDNSDomainName = objRootDomain.Get("canonicalName")
224:        ' Remove the / at the end
225:        strRootDNSDomainName = Mid (strRootDNSDomainName, 1, Len(strRootDNSDomainName) - 1)
...:
230:        ' ------------------------------------------------------------------------------
231:        ' Start the Excel Worksheet reading
232:
233:        ' Bind to an Excel worksheet object
234:        Set objXL = WScript.CreateObject("EXCEL.application")
235:
236:        Wscript.Echo "Reading Excel datasheet '" & strExcelSheetPath & "'"
237:        Wscript.Echo
238:
239:        ' Make it visible
240:        objXL.Visible = True
241:
242:        ' Open the desired workbook containing the users list
243:        objXL.workbooks.open strExcelSheetPath
244:
245:        ' Activate the named page "Export"
246:        objXL.sheets("Export").Activate
247:
248:        ' Put the cursor on the A2 cell
249:        objXL.ActiveSheet.range("A2").Activate
250:
251:        ' Until we run out of rows
252:        Do While objXL.activecell.Value <> ""
253:
254:                ' ------------------------------------------------------------------------
255:                ' Parameters for the user's mailbox creation under LDAP: namespace
256:                strFirstName = objXL.activecell.offset(0,  0).Value
...:
280:                strInfo = objXL.activecell.offset(0, 19).Value
281:
282:                strUserID = CreateUserID (strFirstName, strLastName)
283:
284:                ' Step to the next row to get next user.
285:                objXL.activecell.offset(1, 0).Activate
286:
287:                ' ------------------------------------------------------------------------
```

```
288:                ' Create the user in the Active Directoty
289:                Set objUser = CreateDomainUser ()
290:
291:                ' Create the settings to enable the mailbox for Exchange 2000
292:                If boolEnableEmail Then
293:                   Call EnableEmailAddress (objUser, _
294:                                            strExchangeComputer, _
295:                                            strExchangeOrganization, _
296:                                            strExchangeAdminGroup, _
297:                                            strExchangeStorageGroup, _
298:                                            strExchangeMailboxStore, _
299:                                            CREATE_MB)
300:                End If
...:
307:                ' ------------------------------------------------------------------------
308:                ' Create the home directory for the user on the given server
309:                CreateFolder (strHomeDirRemoteAccess & strUserID)
310:                ' First remove all the rights to clean all ACE entries
311:                Call RemoveRights(strHomeDirRemoteAccess & strUserID, _
312:                                  "REMOVE_ALL_RIGHTS", _
313:                                  FS_OBJECT)
314:                ' Set the rights on the Home directory for "Domain Admins"
315:                Call AddRights(strHomeDirRemoteAccess & strUserID, _
316:                               strAccountNBDomain & "\" & "Domain Admins", _
317:                               ADS_ACETYPE_ACCESS_ALLOWED, _
318:                               ADS_RIGHT_GENERIC_ALL, _
319:                               OBJECT_INHERIT_ACE Or _
320:                               CONTAINER_INHERIT_ACE, _
321:                               FS_OBJECT)
322:
323:                ' Set the rights on the Home directory for the user himself
324:                Call AddRights(strHomeDirRemoteAccess & strUserID, _
325:                               strAccountNBDomain & "\" & strUserID, _
326:                               ADS_ACETYPE_ACCESS_ALLOWED, _
327:                               FILE_READ_DATA Or _
...:
340:                               FILE_READ_CONTROL Or _
341:                               FILE_SYNCHRONIZE, _
342:                               OBJECT_INHERIT_ACE Or _
343:                               CONTAINER_INHERIT_ACE, _
344:                               FS_OBJECT)
345:                ' Remove 'Everyone' Full Control from the REMOVE_ALL_RIGHTS
346:                Call RemoveRights(strHomeDirRemoteAccess & strUserID, _
347:                                  "Everyone", _
348:                                  FS_OBJECT)
349:
350:                ' ------------------------------------------------------------------------
351:                ' Create a profile directory to the name of the created user.
352:                CreateFolder (strProfileDirRemoteAccess & strUserID)
353:                ' First remove all the rights to clean all ACE entries
354:                Call RemoveRights(strProfileDirRemoteAccess & strUserID, _
355:                                  "REMOVE_ALL_RIGHTS", _
356:                                  FS_OBJECT)
357:                ' Set the rights on the Profile directory for "Domain Admins"
358:                Call AddRights(strProfileDirRemoteAccess & strUserID, _
359:                               strAccountNBDomain & "\" & "Domain Admins", _
360:                               ADS_ACETYPE_ACCESS_ALLOWED, _
361:                               ADS_RIGHT_GENERIC_ALL, _
362:                               OBJECT_INHERIT_ACE Or _
363:                               CONTAINER_INHERIT_ACE, _
364:                               FS_OBJECT)
365:                ' Set the rights on the Profile directory for the user himself
366:                Call AddRights(strProfileDirRemoteAccess & strUserID, _
367:                               strAccountNBDomain & "\" & strUserID, _
```

```
368:                                ADS_ACETYPE_ACCESS_ALLOWED, _
369:                                FILE_READ_DATA Or _
...:
382:                                FILE_READ_CONTROL Or _
383:                                FILE_SYNCHRONIZE, _
384:                                OBJECT_INHERIT_ACE Or _
385:                                CONTAINER_INHERIT_ACE, _
386:                                FS_OBJECT)
387:                ' Remove 'Everyone' Full Control from the REMOVE_ALL_RIGHTS
388:                Call RemoveRights(strProfileDirRemoteAccess & strUserID, _
389:                                  "Everyone", _
390:                                  FS_OBJECT)
391:
392:                ' ---------------------------------------------------------------------------
393:                ' Create the share for the Home Directory
394:                CreateShare strAccountComputer, _
395:                            strUserID & "$", _
396:                            strHomeDirLocalAccess & strUserID, _
397:                            2, _
398:                            "Home directory share for " & strDisplayName
399:
400:        Loop
...:
408:        Wscript.Echo vbCRLF & "Operation completed."
409:
410:        Wscript.Quit (0)
411:
412:        ' ------------------------------------------------------------------------------------
413:        Public Function CreateDomainUser ()
...:
422:                ' ---------------------------------------------------------------------------
423:                ' Create the user in the given Domain
424:                strUserPath = "LDAP://" & strAccountComputer & _
425:                              "/CN=Users," & strAccountDomain
426:                Set objUserContainer = GetObject(strUserPath)
427:
428:                strDisplayName = UCase (strLastName) & " " & strFirstName
429:
430:                ' Continue even if the group doesn't exists
431:                On Error Resume Next
432:                ' Clear any prior error to avoid confusion during next test
433:                Err.Clear
434:
435:                Set objUser = objUserContainer.GetObject("user", "CN=" & strDisplayName)
436:                If Err.Number = LDAP_NO_SUCH_OBJECT Then
437:                   ' Create a user in the Domain
438:                   Set objUser = objUserContainer.Create("user", "CN=" & strDisplayName)
439:
440:                   objUser.Put "sAMAccountName", strUserID
441:                   ' Create the account with the minimal set
442:                   objUser.SetInfo
443:                   strEndMessage = "Successfully created Account for user '" & _
444:                                                              strDisplayName & "'."
445:                Else
446:                   strEndMessage = "Successfully updated Account for user '" & _
447:                                                              strDisplayName & "'."
448:                End If
449:
450:                ' Reset the error to no handler
451:                On Error Goto 0
452:
453:                ' Associates properties to the just created account
454:                objUser.Put "userPrincipalName", strUserId & "@" & strRootDNSDomainName
455:                objUser.Put "sn", strLastName
```

```
456:                objUser.Put "givenName", strFirstName
457:                objUser.Put "displayName", strDisplayName
458:                If Len(strDescription) <> 0 Then objUser.Put "description", strDescription
459:
460:                objUser.Put "homeDirectory", "\\" & strAccountComputer & _
461:                            "\" & strUserID & "$"
462:                objUser.Put "homeDrive", cHomeDrive
463:
464:                ' Those properties are not mandatory.
465:                If Len(strTitle) <> 0 Then objUser.Put "title", strTitle
...:
483:                If Len(strMobileNumber) <> 0 Then objUser.Put "mobile", strMobileNumber
484:                If Len(strFaxNumber) <> 0 Then _
485:                                     objUser.Put "facsimileTelephoneNumber", strFaxNumber
486:
487:                If Len(strInfo) <> 0 Then objUser.Put "info", strInfo
488:
489:                objUser.Put "profilePath", "\\" & strAccountComputer & _
490:                            cProfileShare & strUserID
491:                objUser.Put "scriptPath", cLogonScript
492:
493:                ' Get the current Account Control value
494:                intAccountControl = objUser.get ("userAccountControl")
495:                ' Turn off the Disable Account bit to enable the account.
496:                intAccountControl = intAccountControl And (NOT ADS_UF_ACCOUNTDISABLE)
497:                ' Set the value back to AD to enable account.
498:                objUser.put "userAccountControl", CLng (intAccountControl)
499:
500:                ' First password = password
501:                objUser.SetPassword cDefaultPassword
502:
503:                ' Force password change at next logon
504:                objUser.put "pwdLastSet", CLng(0)
505:
506:                ' Commit all the complementary data
507:                objUser.SetInfo
508:
509:                Wscript.Echo
510:                Wscript.Echo strEndMessage
511:
512:                Set CreateDomainUser = objUser
...:
520:        End Function
521:
522:        ' ------------------------------------------------------------------------------
523:        Public Function CreateUserID (strFirstName, strLastName)
...:
527:               strTemp = EliminateSpaces (Lcase(strFirstName) & "." & LCase(strLastName))
528:
529:               If Len (strTemp) > 20 Then strTemp = Left (strTemp, 20)
530:
531:               CreateUserID = strTemp
532:
533:        End Function
534:
535:        </script>
536:</job>
```

The first part of the script (lines 11 to 25) includes all the needed functions. The line 16 includes the EnableE-MailFunction(ADSI).vbs to create the Exchange 2000 mailbox via the ADSI method (See Sample 20 on page 59). This include statement can be changed to include the EnableE-MailFunction(CDOEXM).vbs to use the CDOEXM method (See Sample 21 on page 65). Note that the include statement of the SetSDFunction.vbs at line 22 is mandatory if the script uses the ADSI method.

Lines 39 to 67 define all variable parameters from the Windows 2000 installation to allow the script usage with another Windows 2000 installation:

- Domain Controller for Account (line 43)
- Exchange 2000 Server (line 45)
- Organization Name (line 47)
- Administrative Group (line 48)
- Storage Group (line 49)
- Mailbox Private Store (line 50)
- Logon script name (line 53)
- Default password set at user creation (line 54)
- Home drive letter (line 55)
- Home directory main path (line 61)
- Miscellaneous shares for the Home directory and user Profiles (lines 63 and 64)

Next, the script prompts the user for some default parameters. The presence of Exchange 2000 and the Organization name are checked at line 199 by reusing the "VerifyMSExchangeFunction.vbs" function (included at line 15).

Then, after retrieving some domain characteristics, the script enters the main loop for user creation (lines 252 to 400). First, the script reads the Excel sheet to get all the user parameters for creation (line 256 to 282). Note at line 282, the UserID creation calls a function CreateUserID (line 523). This function is based on a reusable function (EliminateSpaces) included in the "EnableE-MailFunction (ADSI).vbs" include file. The operations in the loop are completed in the following order:

1. Read a row of data from the Excel sheet (lines 256 to 282)
2. Create the user object (line 289)
3. Create the Exchange 2000 Mailbox (lines 292 to 300)
4. Create the Home directory folder (line 309)
5. Set access rights for the Home directory folder (lines 309 to 348):
   - Remove any existing rights defined by the directory creation process (inheritance from the parent directory) at lines 311 to 313.
   - Set full access rights for the "Domain Admins" group (lines 315 to 321)
   - Change access rights for the created user (lines 324 to 344)
   - Remove everyone access ACE from the ACL (lines 346 to 348)
6. Create the Profile directory folder (line 352)
7. Set access right for the Profile directory folder (lines 354 to 390):
   - Remove any existing rights defined by the directory creation process (inheritance from the parent directory) at lines 354 to 356.
   - Set full access rights for the "Domain Admins" group (lines 358 to 364)
   - Change access rights for the created user (lines 366 to 386)
   - Remove everyone access ACE from the ACL (lines 388 to 390)
8. Create the share on the home directory (lines 394 to 398).

This describes the core logic of the script.

The user object creation

The script creates users in the default container named 'Users'. The script tries to bind to the user in this container (lines 424 to 426). If the user does not exist, it creates the object in the container (lines 436 to 445).

Note: A possible script enhancement is the inclusion of the Organization Unit where the user must be created.

The creation process also initiates the mandatory attributes of a user object, such as **sAMAccountName**. **sAMAccountName** must be initiated (line 440) before committing the changes to the directory (line 442).

sAMAccountName: The **sAMAccountName** attribute is the logon name used to support clients and servers from a previous version of Windows (such as Windows NT 4.0, Windows 95, Windows 98, and LAN Manager). Note that the **sAMAccountName** should be less than 20 characters to support these clients and servers. (See CreateUserID function at line 523). The **sAMAccountName** must be unique among all security principal objects within the domain.

Next, an important phase in the user object creation is the assignment of all other object attributes (lines 454 to 507) such as:

userPrincipalName: The User Principal Name (UPN) is an Internet-style login name for the user based on the Internet standard RFC 822. It is shorter than the distinguished name and easier to remember. By convention, this should map to the user's e-mail name. The point of the UPN is to consolidate the e-mail and logon namespaces so that the user needs only to remember a single name. The UPN is constructed at line 454 by using the UserID created at line 282 and the Root DNS name.

sn: This is the family name or last name (line 455).

givenName: This is the first name of the user (line 456).

displayName: This is the name displayed in the address book. This is usually the combination of the users first name, middle initial, and last name (line 428). The assignment is made at line 457.

description: This attribute contains a description for the user object. This attribute can be empty in the Excel sheet because the script tests the presence of a value for the variable strDescription (line 458).

homeDirectory: The **homeDirectory** attribute specifies the path of the home directory for the user. If **homeDrive** is not set, **homeDirectory** should be a local path. If **homeDrive** is set, **homeDirectory** should be a UNC (lines 460 and 461).

homeDrive: The **homeDrive** attribute specifies the drive letter used to map the UNC path specified by **homeDirectory (**line 462).

profilePath: The **profilePath** attribute specifies a path to the user's profile. This value can be a local absolute path, or a UNC path. On line 489, the **profilePath** is composed by the Account computer name, the root share name (i.e. Profile$) and the UserID constructed at line 282.

scriptPath: The scriptPath is the logon script name to be used by the user at logon time (line 491).

From lines 494 to 498, the script sets some default security settings. First, the script reads (line 494) the **userAccountControl** attribute to enable (line 496) the created user. Once enabled, the script replaces the new **userAccountControl** value (line 498).

userAccountControl: The ADSI user object exposes the **AccountDisabled** property. By setting this property to False, it is possible to enable the user and perform the same operation as the lines 494 to 498. Behind the scene, the **AccountDisabled** property modifies the **userAccountControl** attribute. The **userAccountControl** attribute specifies flags that control password, lockout, disable/enable, script, and home directory behavior for the user. Working with the **userAccountControl** attribute allows the control of the different

flags not exposed by ADSI. See Microsoft Active Directory Service Interfaces SDK for more information about the miscellaneous flag values for the **userAccountControl** attribute.

A default password defined at line 54 is set (line 501) and has to be changed at next logon time (line 504).

pwdLastSet: The default value is 0. Zero means the user must change the password at next logon. The value -1 means the user does not need to change the password at next logon. The system sets this value to -1 after the user has set the password.

On the other hand, the following attributes can be empty in the Excel sheet because the script verifies if there is content. (lines 465 to 487):

title	st
department	postalCode
company	c
physicalDeliveryOfficeName	homePhone
telephoneNumber	pager
wWWHomePage	mobile
streetAddress	facsimileTelephoneNumber
postOfficeBox	info
l	

Note: With ADSI, it is important to set an attribute with a variable containing a value, otherwise, an error will be returned to the script.

Creating directories with the File System object

To manipulate the File System object from Windows Script Host, the script must create an instance of the File System Object (line 16). The File System object is part of the Scripting Run-Time library (SCRRUN.DLL). The Scripting Run-Time library was already used to instanciate the Dictionary Object (See on page 12 for more details about the Dictionary Object and see Microsoft Visual Basic Scripting/Java Scripting on-line Documentation for details about the File System object.)

Note: Within the File System Object, several objects are available such as Drive, File, Folder and TextStream.

Once the File System Object instance is created (line 16), the object offers a method to check the presence of a given folder (line 19). The only parameter needed to invoke this method is the folder path. Note that the folder path can be an absolute path or UNC path. In the current scripting case, the caller (Sample 22) will pass a UNC path (lines 309 and 352). Based on the result of test for the folder's existence, the script will create the new folder (line 22) by invoking the CreateFolder method. Sample 23 below is very easy to understand.

Sample 23 Creating the directories for the home directory and the profile directory

```
 1:' VB Script creating folder on the File System if folder does not exist          '
 2:'                                                                                '
 3:' Version 1.00 - Alain Lissoir                                                   '
 4:' Compaq Computer Corporation - Professional Services - Belgium -                '
 5:'                                                                                '
 6:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 7:
 8:Option Explicit
 9:
10:' ---------------------------------------------------------------------------------------
11:Private Sub CreateFolder (strFolderName)
12:
13:Dim objFileSystem
```

```
14:Dim objFolder
15:
16:        Set objFileSystem = Wscript.CreateObject ("Scripting.FileSystemObject")
17:
18:        ' Take care about the existance of the Homedirectory subfolder
19:        If objFileSystem.FolderExists (strFolderName) Then
20:           Wscript.Echo "  Directory '" & strFolderName & "' already exists ..."
21:        Else
22:           Set objFolder = objFileSystem.CreateFolder (strFolderName)
23:           Wscript.Echo "  Successfully created directory '" & strFolderName & "'"
24:
25:           WScript.DisconnectObject objFolder
26:           Set objFolder = Nothing
27:        End If
28:
29:        WScript.DisconnectObject objFileSystem
30:        Set objFileSystem = Nothing
31:
32:End Sub
```

The share object creation

A share creation must be processed in the WinNT: namespace and not in the LDAP: namespace. This is due to the location of the share object inside a service object called LanManServer. The LanManServer object is only accessible from the WinNT: namespace because it is not related to the Active Directory. This section provides a practical example of how the namespace influences the objects/methods available.

Note: See Part 1: "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000", APPENDIX A to know which objects/methods are available for the namespace used.

Creating a share does not require any reference to the File System Object. As usual for an object creation, the script must bind to the object container (that is, the parent of the new object you are creating).

The path of the LanManServer object service is: WinNT://MachineName/LanManServer (line 23). Once you bind to the parent, the script can bind to the share object to check its existence (line 31). Based on the result, the script will create the share object (line 34) or not. Before committing the changes in the WinNT: namespace, it is mandatory for a share creation to set some properties:

- The absolute path of the share location
- The maximum user count of the share
- The share description

This operation is completed at line 36, 37 and 38. To create a hidden share, the caller (Sample 22 above) must append a '$' character to the end of the share name (see line 395 in Sample 22).

Sample 24 Creating the share for the home directory

```
 1:' VB Script creating a File System share via the WinNT: namespace if the given '
 2:' share name does not exist.                                                    '
 3:'                                                                               '
 4:' Version 1.00 - Alain Lissoir                                                  '
 5:' Compaq Computer Corporation - Professional Services - Belgium -               '
 6:'                                                                               '
 7:' Any comments or questions:                    EMail:alain.lissoir@compaq.com '
 8:
 9:Option Explicit
10:
11:Const WINNT_SHARE_NOT_EXIST = &h80070906
12:
```

```
  13:' ------------------------------------------------------------------------------------------
  14:Private Sub CreateShare (strComputer, strShareName, _
  15:                         strDirLocalAccess, strShareCount, _
  16:                         strShareDescription)
  17:
  18:        ' Add the home directory share for the user on the given server
  ..:
  23:        strSharePath = "WinNT://" & strComputer & "/LanManServer"
  24:        Set objShareSrv = GetObject(strSharePath)
  25:
  26:        ' Continue even if the share doesn't exists
  27:        On Error Resume Next
  28:        ' Clear any prior error to avoid confusion during next test
  29:        Err.Clear
  30:
  31:        Set objShare = objShareSrv.GetObject ("FileShare", strShareName)
  32:        If Err.Number = WINNT_SHARE_NOT_EXIST Then
  33:           On Error Goto 0
  34:           Set objShare = objShareSrv.Create ("FileShare", strShareName)
  35:
  36:           objShare.Path = strDirLocalAccess
  37:           objShare.MaxUserCount = strShareCount
  38:           objShare.Description = strShareDescription
  39:
  40:           objShare.SetInfo
  41:
  42:           Wscript.Echo "  Successfully created Directory Share '\\" & _
  43:                        strComputer & "\" & strShareName & "'."
  44:        Else
  45:           Wscript.Echo "  Directory Share '\\" & strComputer & _
  46:                        "\" & strShareName & "' already exists."
  47:        End If
  ..:
  55:End Sub
```

Here is sample output when running Sample 22 (Creating users under Windows 2000 with their associated Exchange 2000 mailboxes) on page 68.

```
Reading Excel datasheet 'C:\Data\Excel\PopulateUsers.xls'

Found Exchange Organization called 'First Organization'.

Successfully created Account for user 'WEST James'.
  Successfully created Microsoft Exchange 2000 Mailbox for user 'WEST James'.
  Security Descriptor for MB object initialized.
  Successfully created directory '\\MYW2KDC\HOME$\james.west'
  All rights are removed, Trustee 'Everyone' (Full Control) has been added.
  Trustee 'MYW2KDOMAIN\Domain Admins' has been added to '\\MYW2KDC\HOME$\james.west'
  Trustee 'MYW2KDOMAIN\james.west' has been added to '\\MYW2KDC\HOME$\james.west'
  Trustee 'Everyone' has been removed from '\\MYW2KDC\HOME$\james.west'
  Successfully created directory '\\MYW2KDC\Profile$\james.west'
  All rights are removed, Trustee 'Everyone' (Full Control) has been added.
  Trustee 'MYW2KDOMAIN\Domain Admins' has been added to '\\MYW2KDC\Profile$\james.west'
  Trustee 'MYW2KDOMAIN\james.west' has been added to '\\MYW2KDC\Profile$\james.west'
  Trustee 'Everyone' has been removed from '\\MYW2KDC\Profile$\james.west'
  Successfully created Directory Share '\\MYW2KDC\james.west$'.
```

CONCLUSION

Windows Script Host under Windows 2000 offers a lot of new possibilities. The real limits come from the number of reusable COM objects available in the scripting environment. As discovered in Part 1: "Understanding the Microsoft Windows Script Host and the Active Directory Service Interfaces in Windows 2000", the challenge lies mainly in knowing what all the COM objects are available in a Windows 2000 system.

Not only existing COM interfaces can be reused, but it is also possible to construct "scripted COM objects" via XML encapsulation. This powerful feature offers scripting code reusability to other scripting languages but also to other automation languages such as Visual Basic, Perl or Java.

By offering WSH to developers and administrators, Microsoft provides a powerful tool, usable from the command line. This lets Windows handle scripting operations in a similar way as a Unix Shell.

In fact, WSH goes further than a simple scripting language because even if the basic engine is a scripting engine (VB Script or Java Script), complex operations require clear structure and organization of code, variables, functions and miscellaneous components, just as they do in the real programming world.

The frontier between both worlds is seriously reduced because both can make use of each other. In the past, a script generally started many applications in a clearly defined way. Now, with these new possibilities, an application can invoke a script to perform some operations. This has advantages, such as being able to change program logic (for example, a parsing procedure) without recompiling the original software, because this logic is in a Windows Script Component.

APPENDIX A: USER OBJECT ATTRIBUTE DEFINITIONS

Table 1 The user object attributes, the syntax properties and the property types.

(Mandatory attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	**Syntax**	**Multi-valued**	**Indexed**	**GC Copy**	**SystemFlags (ADS_SYSTEMFLAG)**
cn	DirectoryString	False	Yes	Yes	CR_NTDS_DOMAIN;
instanceType	INTEGER	False		Yes	CR_NTDS_DOMAIN;
nTSecurityDescriptor	ObjectSecurityDescriptor	False		Yes	CR_NTDS_DOMAIN;
objectCategory	DN	False	Yes	Yes	CR_NTDS_DOMAIN;
objectClass	OID	True		Yes	CR_NTDS_DOMAIN;
objectSid	OctetString	False	Yes	Yes	CR_NTDS_DOMAIN;
sAMAccountName	DirectoryString	False	Yes	Yes	CR_NTDS_DOMAIN;

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	**Syntax**	**Multivalued**	**Indexed**	**GC Copy**	**SystemFlags (ADS_SYSTEMFLAG)**
accountExpires	INTEGER8	False			
accountNameHistory	DirectoryString	True			
aCSPolicyName	DirectoryString	False			
adminCount	INTEGER	False			
adminDescription	DirectoryString	False			
adminDisplayName	DirectoryString	False		Yes	
allowedAttributes	OID	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
allowedAttributesEffective	OID	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
allowedChildClasses	OID	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
allowedChildClassesEffective	OID	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
altRecipient	DN	False		Yes	
altRecipientBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
altSecurityIdentities	DirectoryString	True	Yes	Yes	CR_NTDS_DOMAIN;

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
assistant	DN	False			
attributeCertificate	OctetString	True		Yes	
authOrig	ORName	True		Yes	
authOrigBL	DN	True		Yes	CR_NTDS_NC;ATTR_NOT_REPLICATED;
autoReply	Boolean	False		Yes	
autoReplyMessage	DirectoryString	False			
badPasswordTime	INTEGER8	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
badPwdCount	INTEGER	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
bridgeheadServerListBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
businessRoles	OctetString	False			
c	DirectoryString	False		Yes	CR_NTDS_DOMAIN;
canonicalName	DirectoryString	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
co	DirectoryString	False			
codePage	INTEGER	False			
comment	DirectoryString	False			
company	DirectoryString	False		Yes	
controlAccessRights	OctetString	True			
countryCode	INTEGER	False			
createTimeStamp	GeneralizedTime	False			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
dBCSPwd	OctetString	False			
defaultClassStore	DN	True			
deletedItemFlags	INTEGER	False		Yes	
delivContLength	INTEGER	False		Yes	
deliverAndRedirect	Boolean	False		Yes	
deliveryMechanism	INTEGER	False		Yes	
delivExtContTypes	OctetString	True		Yes	
department	DirectoryString	False		Yes	
description	DirectoryString	True		Yes	
desktopProfile	DirectoryString	False			
destinationIndicator	PrintableString	True			
directReports	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
displayName	DirectoryString	False	Yes	Yes	
displayNamePrintable	PrintableString	False		Yes	
distinguishedName	DN	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
division	DirectoryString	False			
dLMemDefault	INTEGER	False		Yes	
dLMemRejectPerms	ORName	True		Yes	
dLMemRejectPermsBL	DN	True		Yes	CR_NTDS_NC;ATTR_NOT_REPLICATED;
dLMemSubmitPerms	ORName	True		Yes	
dLMemSubmitPermsBL	DN	True		Yes	CR_NTDS_NC;ATTR_NOT_REPLICATED;
dnQualifier	DirectoryString	False			
dSASignature	OctetString	False			
dSCorePropagationData	GeneralizedTime	True		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
dynamicLDAPServer	DN	False			
employeeID	DirectoryString	False			
employeeType	DirectoryString	False			
enabledProtocols	INTEGER	False		Yes	
expirationTime	UTCTime	False	Yes		
extensionAttribute1	DirectoryString	False		Yes	
extensionAttribute10	DirectoryString	False		Yes	
extensionAttribute11	DirectoryString	False		Yes	
extensionAttribute12	DirectoryString	False		Yes	
extensionAttribute13	DirectoryString	False		Yes	
extensionAttribute14	DirectoryString	False		Yes	
extensionAttribute15	DirectoryString	False		Yes	
extensionAttribute2	DirectoryString	False		Yes	
extensionAttribute3	DirectoryString	False		Yes	
extensionAttribute4	DirectoryString	False		Yes	
extensionAttribute5	DirectoryString	False		Yes	
extensionAttribute6	DirectoryString	False		Yes	
extensionAttribute7	DirectoryString	False		Yes	
extensionAttribute8	DirectoryString	False		Yes	
extensionAttribute9	DirectoryString	False		Yes	
extensionData	OctetString	True			
extensionName	DirectoryString	True			
facsimileTelephoneNumber	DirectoryString	False		Yes	
flags	INTEGER	False		Yes	
folderPathname	DirectoryString	False		Yes	

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
formData	OctetString	False			
forwardingAddress	DirectoryString	False			
fromEntry	Boolean	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
frsComputerReferenceBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
fRSMemberReferenceBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
fSMORoleOwner	DN	False	Yes		
garbageCollPeriod	INTEGER	False		Yes	
generationQualifier	DirectoryString	False			
givenName	DirectoryString	False	Yes	Yes	
groupMembershipSAM	OctetString	False			
groupPriority	DirectoryString	True			
groupsToIgnore	DirectoryString	True			
heuristics	INTEGER	False		Yes	
homeDirectory	DirectoryString	False			
homeDrive	DirectoryString	False			
homeMDB	DN	False		Yes	
homeMTA	DN	False		Yes	
homePhone	DirectoryString	False		Yes	
homePostalAddress	DirectoryString	False			
houseIdentifier	DirectoryString	False			
importedFrom	DirectoryString	False	Yes	Yes	
info	DirectoryString	False		Yes	
initials	DirectoryString	False		Yes	
internationalISDNNumber	NumericString	True			
internetEncoding	INTEGER	False		Yes	
ipPhone	DirectoryString	False		Yes	
isCriticalSystemObject	Boolean	False			
isDeleted	Boolean	False		Yes	CR_NTDS_DOMAIN;
isPrivilegeHolder	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
kMServer	DN	False			
l	DirectoryString	False	Yes	Yes	CR_NTDS_DOMAIN;
labeledURI	DirectoryString	True			
language	DirectoryString	False			
languageCode	INTEGER	False			

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
lastKnownParent	DN	False			
lastLogoff	INTEGER8	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
lastLogon	INTEGER8	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
legacyExchangeDN	CaseIgnoreString	False	Yes	Yes	
lmPwdHistory	OctetString	True			
localeID	INTEGER	True			
lockoutTime	INTEGER8	False			
logonCount	INTEGER	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
logonHours	OctetString	False			
logonWorkstation	OctetString	False			
mail	DirectoryString	False	Yes	Yes	
mailNickname	DirectoryString	False	Yes	Yes	
managedObjects	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
manager	DN	False		Yes	
mAPIRecipient	Boolean	False		Yes	
masteredBy	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
maxStorage	INTEGER8	False			
mDBOverHardQuotaLimit	INTEGER	False		Yes	
mDBOverQuotaLimit	INTEGER	False		Yes	
mDBStorageQuota	INTEGER	False		Yes	
mDBUseDefaults	Boolean	False		Yes	
memberOf	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
mhsORAddress	DirectoryString	True			
middleName	DirectoryString	False			
mobile	DirectoryString	False		Yes	
modifyTimeStamp	GeneralizedTime	False			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
mS-DS-ConsistencyChildCount	INTEGER	False			
mS-DS-ConsistencyGuid	OctetString	False			
mS-DS-CreatorSID	OctetString	False	Yes		
msExchADCGlobalNames	DirectoryString	True	Yes	Yes	
msExchALObjectVersion	INTEGER	False			
msExchConferenceMailboxBL	DN	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
msExchControllingZone	DN	False			
msExchCustomProxyAddresses	DirectoryString	True		Yes	

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
msExchExpansionServerName	DirectoryString	False		Yes	
msExchFBURL	DirectoryString	False	Yes	Yes	
msExchHideFromAddressLists	Boolean	False		Yes	
msExchHomeServerName	DirectoryString	False		Yes	
msExchIMACL	OctetString	True		Yes	
msExchIMAddress	PrintableString	False	Yes	Yes	
msExchIMMetaPhysicalURL	PrintableString	False	Yes	Yes	
msExchIMPhysicalURL	PrintableString	False	Yes	Yes	
msExchIMVirtualServer	DN	False	Yes	Yes	
msExchInconsistentState	INTEGER	False			
msExchMailboxGuid	OctetString	False	Yes	Yes	
msExchMailboxSecurityDescriptor	ObjectSecurityDescriptor	False		Yes	
msExchMailboxUrl	DirectoryString	False		Yes	
msExchMasterAccountSid	OctetString	False	Yes	Yes	
msExchPfRootUrl	DirectoryString	False		Yes	
msExchPoliciesExcluded	DirectoryString	True			
msExchPoliciesIncluded	DirectoryString	True			
msExchPolicyEnabled	Boolean	False			
msExchPolicyOptionList	OctetString	True			
msExchPreviousAccountSid	OctetString	False	Yes	Yes	
msExchProxyCustomProxy	DirectoryString	True			
msExchQueryBaseDN	DN	False		Yes	
msExchRecipLimit	INTEGER	False		Yes	
msExchResourceGUID	DirectoryString	True	Yes	Yes	
msExchResourceProperties	DirectoryString	True			
msExchTUIPassword	OctetString	False		Yes	
msExchTUISpeed	INTEGER	False		Yes	
msExchTUIVolume	INTEGER	False		Yes	
msExchUnmergedAttsPt	OctetString	False	Yes		
msExchUseOAB	DN	False		Yes	
msExchUserAccountControl	INTEGER	False	Yes	Yes	
msExchVoiceMailboxID	DirectoryString	False	Yes	Yes	
mSMQDigests	OctetString	True	Yes	Yes	
mSMQDigestsMig	OctetString	True		Yes	

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
mSMQSignCertificates	OctetString	False		Yes	
mSMQSignCertificatesMig	OctetString	False		Yes	
msNPAllowDialin	Boolean	False			
msNPCallingStationID	IA5String	True			
msNPSavedCallingStationID	IA5String	True			
msRADIUSCallbackNumber	IA5String	False			
msRADIUSFramedIPAddress	INTEGER	False			
msRADIUSFramedRoute	IA5String	True			
msRADIUSServiceType	INTEGER	False			
msRASSavedCallbackNumber	IA5String	False			
msRASSavedFramedIPAddress	INTEGER	False			
msRASSavedFramedRoute	IA5String	True			
name	DirectoryString	False	Yes	Yes	CR_NTDS_DOMAIN;
netbootSCPBL	DN	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
networkAddress	CaseIgnoreString	True		Yes	
nonSecurityMemberBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
ntPwdHistory	OctetString	True			
o	DirectoryString	True		Yes	CR_NTDS_DOMAIN;
objectGUID	OctetString	False	Yes	Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
objectVersion	INTEGER	False			
operatorCount	INTEGER	False			
otherFacsimileTelephoneNumber	DirectoryString	True		Yes	
otherHomePhone	DirectoryString	True		Yes	
otherIpPhone	DirectoryString	True		Yes	
otherLoginWorkstations	DirectoryString	True			
otherMailbox	DirectoryString	True			
otherMobile	DirectoryString	True		Yes	
otherPager	DirectoryString	True		Yes	
otherTelephone	DirectoryString	True		Yes	
otherWellKnownObjects	DNWithBinary	True			
ou	DirectoryString	True	Yes	Yes	CR_NTDS_DOMAIN;
pager	DirectoryString	False		Yes	
partialAttributeDeletionList	OctetString	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
partialAttributeSet	OctetString	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
personalPager	DirectoryString	False			
personalTitle	DirectoryString	False			
physicalDeliveryOfficeName	DirectoryString	False	Yes	Yes	
pOPCharacterSet	DirectoryString	False			
pOPContentFormat	DirectoryString	False			
possibleInferiors	OID	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
postalAddress	DirectoryString	True			
postalCode	DirectoryString	False		Yes	
postOfficeBox	DirectoryString	True		Yes	
preferredDeliveryMethod	INTEGER	True			
preferredOU	DN	False			
primaryGroupID	INTEGER	False	Yes	Yes	CR_NTDS_DOMAIN;
primaryInternationalISDNNumber	DirectoryString	False			
primaryTelexNumber	DirectoryString	False			
profilePath	DirectoryString	False			
protocolSettings	DirectoryString	True		Yes	
proxiedObjectName	DNWithBinary	False		Yes	CR_NTDS_DOMAIN;
proxyAddresses	DirectoryString	True	Yes	Yes	
publicDelegates	DN	True		Yes	
publicDelegatesBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
pwdLastSet	INTEGER8	False			
queryPolicyBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
registeredAddress	OctetString	True			
replicatedObjectVersion	INTEGER	False		Yes	
replicationSensitivity	INTEGER	False			
replicationSignature	OctetString	False		Yes	
replPropertyMetaData	OctetString	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
replUpToDateVector	OctetString	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
repsFrom	OctetString	True		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
repsTo	OctetString	True		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
revision	INTEGER	False			
rid	INTEGER	False			
sAMAccountType	INTEGER	False	Yes	Yes	CR_NTDS_DOMAIN;
scriptPath	DirectoryString	False			

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
sDRightsEffective	INTEGER	False			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
secretary	DirectoryString	False			
securityIdentifier	OctetString	False			
securityProtocol	OctetString	True		Yes	
seeAlso	DN	True			
serverReferenceBL	DN	False			CR_NTDS_NC;ATTR_NOT_REPLICATED;
servicePrincipalName	DirectoryString	True	Yes	Yes	CR_NTDS_DOMAIN;
showInAddressBook	DN	True		Yes	
showInAdvancedViewOnly	Boolean	False	Yes		
sIDHistory	OctetString	True	Yes	Yes	CR_NTDS_DOMAIN;
siteObjectBL	DN	True			CR_NTDS_NC;ATTR_NOT_REPLICATED;
sn	DirectoryString	False	Yes	Yes	
st	DirectoryString	False		Yes	CR_NTDS_DOMAIN;
street	DirectoryString	False		Yes	CR_NTDS_DOMAIN;
streetAddress	DirectoryString	False		Yes	
submissionContLength	INTEGER	False		Yes	
subRefs	DN	True		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
subSchemaSubEntry	DN	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
supplementalCredentials	OctetString	True			
supportedAlgorithms	OctetString	False		Yes	
systemFlags	INTEGER	False			
targetAddress	DirectoryString	False	Yes	Yes	
telephoneAssistant	DirectoryString	False			
telephoneNumber	DirectoryString	False		Yes	
teletexTerminalIdentifier	OctetString	True			
telexNumber	OctetString	True			
terminalServer	OctetString	False			
textEncodedORAddress	DirectoryString	False	Yes	Yes	
thumbnailLogo	OctetString	False			
thumbnailPhoto	OctetString	False			
title	DirectoryString	False		Yes	
tokenGroups	OctetString	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
tokenGroupsGlobalAndUniversal	OctetString	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;
tokenGroupsNoGCAcceptable	OctetString	True			DOMAIN_DISALLOW_RENAME;ATTR_IS_CONTRUCTED;

(Optional attributes)

CN=Alain Lissoir	ClassObject=user	AdsPath=LDAP://CN=Alain Lissoir,CN=Users,DC=w2k-home,DC=com			
Attribute Name	Syntax	Multivalued	Indexed	GC Copy	SystemFlags (ADS_SYSTEMFLAG)
unauthOrig	ORName	True		Yes	
unauthOrigBL	DN	True		Yes	CR_NTDS_NC;ATTR_NOT_REPLICATED;
unicodePwd	OctetString	False			
unmergedAtts	OctetString	False	Yes		
url	DirectoryString	True		Yes	
userAccountControl	INTEGER	False	Yes	Yes	CR_NTDS_DOMAIN;
userCert	OctetString	False		Yes	
userCertificate	OctetString	True		Yes	
userParameters	DirectoryString	False			
userPassword	OctetString	True			
userPrincipalName	DirectoryString	False	Yes	Yes	CR_NTDS_DOMAIN;
userSharedFolder	DirectoryString	False			
userSharedFolderOther	DirectoryString	True			
userSMIMECertificate	OctetString	True		Yes	
userWorkstations	DirectoryString	False			
uSNChanged	INTEGER8	False	Yes	Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
uSNCreated	INTEGER8	False	Yes	Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
uSNDSALastObjRemoved	INTEGER8	False			
USNIntersite	INTEGER	False	Yes		
uSNLastObjRem	INTEGER8	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
uSNSource	INTEGER8	False			
versionNumber	INTEGER	False		Yes	
wbemPath	DirectoryString	True			
wellKnownObjects	DNWithBinary	True		Yes	CR_NTDS_DOMAIN;
whenChanged	GeneralizedTime	False		Yes	CR_NTDS_NC;CR_NTDS_DOMAIN;ATTR_NOT_REPLICATED;
whenCreated	GeneralizedTime	False		Yes	
wWWHomePage	DirectoryString	False		Yes	
x121Address	NumericString	True			

APPENDIX B: USER GUI RIGHTS

Note: **Be sure to read the table carefully.** First select a GUI right from the top and by looking down in the same column, a '•' is placed each time the flag (in the left column) has to be used for the selected right. Reading the table in the other direction (starting from the flag) will not create the GUI right at run-time.

	AD Standard Rights																	
	Standard View					Advanced View												
	Full Control	Read	Write	Create All Child Object	Delete All Child Object	Full Control	List Contents	Read All Properties	Write All Properties	Delete	Delete SubTree	Read Permissions	Modify Permissions	Modify Owner	All Validated Rights	All Extended Rights	Create All Child Objects	Delete All Child Objects
ACE Type																		
ADS_ACETYPE_ACCESS_ALLOWED	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ADS_ACETYPE_ACCESS_DENIED																		
ACE AccessMask																		
ADS_RIGHT_GENERIC_READ		•																
ADS_RIGHT_GENERIC_WRITE			•															
ADS_RIGHT_GENERIC_EXECUTE																		
ADS_RIGHT_GENERIC_ALL	•																	
ADS_RIGHT_DELETE	•					•				•								
ADS_RIGHT_READ_CONTROL	•	•				•						•						
ADS_RIGHT_WRITE_DAC	•					•							•					
ADS_RIGHT_WRITE_OWNER	•					•								•				
ADS_RIGHT_SYNCHRONIZE																		
ADS_RIGHT_ACCESS_SYSTEM_SECURITY																		
ADS_RIGHT_DS_CREATE_CHILD	•			•		•											•	
ADS_RIGHT_DS_DELETE_CHILD	•				•	•												•
ADS_RIGHT_ACTRL_DS_LIST	•	•				•	•											
ADS_RIGHT_DS_SELF	•		•			•									•			
ADS_RIGHT_DS_READ_PROP	•	•				•		•										
ADS_RIGHT_DS_WRITE_PROP	•		•			•			•									
ADS_RIGHT_DS_DELETE_TREE	•					•					•							
ADS_RIGHT_DS_LIST_OBJECT	•					•												
ADS_RIGHT_DS_CONTROL_ACCESS	•					•									•	•		

Table 2 The Active Directory standard GUI rights and their associated flag values.

	FS Rights																	
	Standard View						Advanced View											
	Full Control	Modify	Read & Execute	List Folder Contents	Read	Write	Traverse Folder / Execute File	List Folder / Read Data	Read Attributes	Read Extended Attributes	Create Files / Write Data	Create Folders / Append Data	Write Attributes	Write Extended Attributes	Delete SubFolders and Files	Delete	Read Permissions	Change Permissions	Take Ownership
ACE Type																			
ACCESS_ALLOWED_ACE_TYPE	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ACCESS_DENIED_ACE_TYPE																			
ACE AccessMask																			
FILE_ALL_ACCESS	•																		
FILE_GENERIC_READ					•														
FILE_GENERIC_WRITE						•													
FILE_GENERIC_EXECUTE			•	•															
FILE_READ_DATA	•	•	•	•	•			•											
FILE_LIST_DIRECTORY	•	•	•	•	•			•											
FILE_WRITE_DATA	•	•				•					•								
FILE_ADD_FILE	•	•				•					•								
FILE_APPEND_DATA	•	•				•						•							
FILE_ADD_SUBDIRECTORY	•	•				•						•							
FILE_READ_EA	•	•	•	•	•					•									
FILE_WRITE_EA	•	•				•								•					
FILE_EXECUTE	•	•	•	•			•												
FILE_TRAVERSE	•	•	•	•			•												
FILE_DELETE_CHILD	•														•				
FILE_READ_ATTRIBUTES	•	•	•	•	•				•										
FILE_WRITE_ATTRIBUTES	•	•				•							•						
FILE_DELETE	•	•														•			
FILE_READ_CONTROL	•	•	•	•	•												•		
FILE_WRITE_DAC	•																	•	
FILE_WRITE_OWNER	•																		•
FILE_SYNCHRONIZE	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 3 The File System GUI rights and their associated flag values.

	This object only	This object and all child objects	Child objects only
ACE Flags			
OBJECT_INHERIT_ACE			
CONTAINER_INHERIT_ACE		•	•
NO_PROPAGATE_INHERIT_ACE			
INHERIT_ONLY_ACE			•
INHERITED_ACE			
VALID_INHERIT_FLAGS			

Table 4 The Active Directory GUI inheritance and their associated flag values.

	This Folder Only	This Folder, subfolders and files	This folder and subfolders	This folder and files	Subfolders and files only	Subfolder only	Files only
ACE Flags							
OBJECT_INHERIT_ACE		•		•	•		•
CONTAINER_INHERIT_ACE		•	•		•	•	
NO_PROPAGATE_INHERIT_ACE							
INHERIT_ONLY_ACE					•	•	•
INHERITED_ACE							
VALID_INHERIT_FLAGS							

Table 5 The File System GUI inheritance and their associated flag values.

	AD Exchange 2000 Rights											
	Standard View						Advanced View					
	Delete Mailbox Storage	Read Permissions	Change Permissions	Take Ownership	Full Mailbox Access	Associated External Account	Delete Mailbox Storage	Read Permissions	Change Permissions	Take Ownership	Full Mailbox Access	Associated External Account
ACE Type												
ADS_ACETYPE_ACCESS_ALLOWED ADS_ACETYPE_ACCESS_DENIED	•	•	•	•	•	•	•	•	•	•	•	•
ACE AccessMask												
RIGHT_MB_FULL_MB_ACCESS					•	•					•	•
RIGHT_MB_SEND_AS	•	•	•	•	•	•	•	•	•	•	•	•
RIGHT_MB_EXTERNAL_ACCOUNT						•						•
RIGHT_MB_DELETE	•						•					
RIGHT_MB_READ_PERMISSIONS		•						•				
RIGHT_MB_CHANGE_PERMISSIONS			•						•			
RIGHT_MB_TAKE_OWNERSHIP				•						•		

Table 6 The Exchange 2000 GUI rights and their associated flag values.

Warning: The values presented in Table 6 above do not come from the SDK. At the time of publication, no information was available about which flag values to apply to enable any specific rights. The values presented here were found by deciphering each ACE entry set via the GUI (in other words, by trial and error).

	This object only	Inherit only	This Object and Subcontainers	This object and children objects	Subcontainers only	Children object only	This object, subcontainers and children objects	Subcontainers and children objects
ACE Flags								
OBJECT_INHERIT_ACE				•		•	•	•
CONTAINER_INHERIT_ACE			•		•		•	•
NO_PROPAGATE_INHERIT_ACE								
INHERIT_ONLY_ACE		•			•	•		•
INHERITED_ACE								

Table 7 The Exchange 2000 GUI inheritance and their associated flag values.