COMPAQ

WHITE PAPER

technical level

BASIC ADVANCED

October 2000

Prepared by
Alain Lissoir

Technology Consultant
Compaq Services

Compaq Computer
Corporation

Understanding Microsoft Windows Script Host and Active Directory Service Interfaces in Windows 2000

This white paper gives an overview of the scripting possibilities offered by the Windows Script Host (WSH) and Active Directory Service Interface (ADSI) components under Windows 2000. It explains the basic concepts and practices related to WSH and ADSI.

After a WSH architectural overview, the document explains how to use this technology to access basic Windows system components. This paper also provides a general ADSI architectural overview and explains the basic access methods for the Active Directory. Through several simple examples, the reader will discover how to combine WSH functions with ADSI components in order to build a Logon Script under Windows 2000.

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

October 2000

2nd version.

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

TABLE OF CONTENTS

INTRODUCTION 6

PREREQUISITES 8

WHAT'S NEW IN THIS UPDATED VERSION? 9

WINDOWS SCRIPT HOST 10

DESCRIPTION 10
THE OBJECT MODEL 11
THE WINDOWS SCRIPT HOST CAPABILITIES 14

USING BASIC WSH FUNCTIONS IN A WINDOWS 2000 LOGON SCRIPT 15

PROVIDING INFORMATION ON THE SCRIPT ENVIRONMENT AND THE WINDOWS SYSTEM 15
READING ARGUMENTS FROM THE COMMAND LINE 16
MANIPULATING THE ENVIRONMENT VARIABLES 18
CREATING, RETRIEVING AND DELETING NETWORK CONNECTIONS 19
RUNNING EXTERNAL PROGRAMS 20
CREATING SHORTCUTS 21
ACCESSING THE REGISTRY 21
USING REGISTERED COM OBJECTS 23
ACCESSING THE FILE SYSTEM 24
SUPPORT FOR INCLUDE FILES 25
SUPPORT FOR MULTIPLE ENGINES 26
PAUSING A SCRIPT 26
DRAG AND DROP SUPPORT 26
STANDARD INPUT AND STANDARD OUTPUT 26

ACTIVE DIRECTORY SERVICE INTERFACES 28

DESCRIPTION 28
ADSI IMPLEMENTATION 30
ADSI OBJECTS AND NAMESPACES 31

HOW IS THE ACTIVE DIRECTORY STRUCTURED? 33

THE ROOTDSE 33
THE NAMING CONTEXTS 35
The Schema Naming Context *36*
The Configuration Naming Context *36*
The Domain Naming Context *36*
The Default Naming Context *36*
The Root Naming Context *36*

ACCESSING THE DIRECTORY 37

THE BINDING OPERATION 37
The Namespace *37*
Using other credentials *37*
THE DISTINGUISHED NAME 38
WHICH SYNTAX TO USE? 39

The Namespace syntax differences *39*
The object syntax differences *41*
Discovering the syntax of an object *41*
Discovering the Directory Tree, Objects, attributes and syntaxes *43*
Getting and setting object's attribute values in respect of the syntax *45*
Which methods to use to manipulate ADSI data? *47*

COMPLETING THE WSH FUNCTIONS SET WITH ADSI INFORMATION 50

GETTING THE USER'S DISTINGUISHED NAME 50
GETTING THE USER FULLNAME 51
GROUP MEMBERSHIP CHECKING 52
GETTING THE DEFAULT DOMAIN OR THE ROOT DOMAIN DISTINGUISHED NAME 53
LOGON SCRIPT SAMPLE 54

CONCLUSION 55

APPENDIX A: ADSI INTERFACES LIST 56

APPENDIX B: REFERENCES AND POINTERS 58

TABLE OF SAMPLES

Sample 1 Providing information on the running script and Windows System. *15*
Sample 2 Reading arguments from the command line *17*
Sample 3 Manipulating the environment variables *18*
Sample 4 Creating, retrieving and deleting network connections *19*
Sample 5 Manipulating Desktop objects *21*
Sample 6 Accessing the registry *21*
Sample 7 Using registered COM objects *23*
Sample 8 Accessing the file system *24*
Sample 9 Standard Input and Standard Output usage *26*
Sample 10: Looking at some RootDSE object's attributes *34*
Sample 11: Getting Administrator's group membership from a Windows 2000 DC in different namespaces. *39*
Sample 12: Accessing the syntax of an attribute *41*
Sample 13: Loading the entire Active Directory Tree objects with their attributes and syntaxes loaded in an Excel sheet. *43*
Sample 14 Getting the user distinguished name *50*
Sample 15 Information available from the IADsSsystemInfo interface. *51*
Sample 16 Getting the user Fullname *51*
Sample 17 Group Membership checking *52*
Sample 18 Getting the Default Domain or the Root Domain distinguished name *53*
Sample 19 Logon Script Sample functions overview *54*

TABLE OF FIGURES

Figure 1 Microsoft Scripting architecture *10*
Figure 2 The Wscript object *11*
Figure 3 The Wscript.Shell object *12*
Figure 4 The WshNetwork object *13*
Figure 5 ADSI Service Providers *29*
Figure 6 ADSI Architecture and language support. *30*

Figure 7 ADSI Interfaces for each name space used. *56*

INTRODUCTION

Since the introduction of the PC to the world in 1981-1982, there has been a need to automate certain system functions. From the simple .bat (batch file) up to the most complex tasks, administrators and software developers have always wanted a way to describe and automate common tasks. In 1982, the purpose may have been to automate the simple compilation of a program developed in IBM/Microsoft Basic 1.0 or to simply make a safe copy of files located on the 10MB hard disk to the famous 180K/360K 5” ¼ floppies.

The evolution of the PC operating system has consistently proven that the interpreted batch file from the COMMAND.COM was really not enough to address all the needs.

During all those years, due to this lack of support in the operating system, third party vendors have led many initiatives to solve the scripting problem. One of the best examples is the adaptation of Unix tools to DOS (i.e. GNU Utilities, Perl Scripting). This was a great help to many administrator and developers. But the challenge was to choose the right tools, and sometimes the right script interpreter.

One important initiative came from IBM with the arrival of OS/2 in 1988. IBM delivered an established scripting language called REXX. As part of the OS/2 Operating System, REXX offered all the functions required to achieve common tasks such as automating a full Operating System setup, installing applications, creating icons, calling 32 bits API DLLs and so on. IBM developed a DOS version of REXX that was available in PC DOS v7.0.

In addition to the “classical” .bat extension, OS/2 added a new file type for native OS/2 command files and REXX script files. The extension was called .CMD (Command File).

Of course, under Unix, such scripting capabilities were certainly not new. The Korn Shell, Bourn Shell, and C Shell, had all offered similar features for quite some time.

More or less in parallel with OS/2 came the arrival of Windows 3.1. Windows 3.1, which was still based on DOS, offered no real scripting language and the REXX support under Windows 3.1 was not well integrated. Scripting continued to be neglected in the DOS world. Most people expected many new scripting features with the arrival of the first Windows NT version. In fact, there were some new builtin features available in Windows NT 4.0 Service Pack 3 through the use of .CMD files. Unfortunately, those new features fall far short of the capabilities available under a Unix Shell or a REXX command file.

Today, with the arrival of Windows 2000, those waiting since 1982 for a valid and powerful scripting language built into the Operating System will receive an answer. Windows Script Host (WSH) is the Microsoft answer.

WSH fills the gap between the classical batch file programming and the real advanced programming world.

Microsoft does not restrict WSH to Windows 2000. It is also available for Windows NT 4.0 SP3. WSH version 1.0 (called Windows Scripting Host) is included in the Windows NT 4.0 Option Pack; WSH version 2.0 (called Windows Script Host) is available from the web.

This white paper provides a brief overview of WSH and the Active Directory Service Interface (ADSI). The paper is divided into two parts:

1. The first part (this document) describes both technologies and discusses the basic functions.

2. The second part goes further. Explaining how WSH and ADSI make an Administrator’s life easier through practical scripting samples.

The capabilities described in this white paper are based on Windows 2000. Some changes may occur with later builds.

PREREQUISITES

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

- VisualBasic Scripting
- How to use COM objects inside VisualBasic Scripting
- Windows 2000 Active Directory
- Windows 2000 concepts

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

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

WHAT'S NEW IN THIS UPDATED VERSION?

APPENDIX B on page 58 contains updated references and pointers.

WINDOWS SCRIPT HOST

Description

Windows Script Host is an infrastructure to run scripts. The implementation uses callable ActiveX interfaces. This is a language-independent host allowing running scripts under Windows 9x, Windows NT and Windows 2000. When executing a script for Windows Script Host, it can be run in either of two modes:

- Command line mode (Cscript.EXE)

  This engine will display the output as a regular batch file in a DOS Command prompt (Cscript as Command Script). The default engine can be determined from the command line:

  Cscript.Exe //H:Cscript

- Window mode (Wscript.EXE)

  Instead of displaying messages in a DOS Command prompt, this engine will create a message popup window to display each message (Wscript as Window script). The default engine can be determined from the command line:

  Wscript.Exe //H:Wscript

You can execute scripts with either engine; the results will be the same. However, running a script in window mode will require user intervention each time a message is displayed. When invoking a script from an interactive application, Wscript.Exe is a good choice. For a logon script, Cscript.Exe is usually better.

Windows Script Host supports both the Visual Basic and Java script languages but the architecture permits to extend this support to any other language by developing new ActiveX (i.e. Perl). At this time, two separate versions of Windows Script Host exist. Each version handles different scripting languages slightly differently. Both versions are structured around the scripting ActiveX interfaces. This means that Windows Script Host reuses scripting components already used by other existing applications such as Internet Explorer or Internet Information Server.

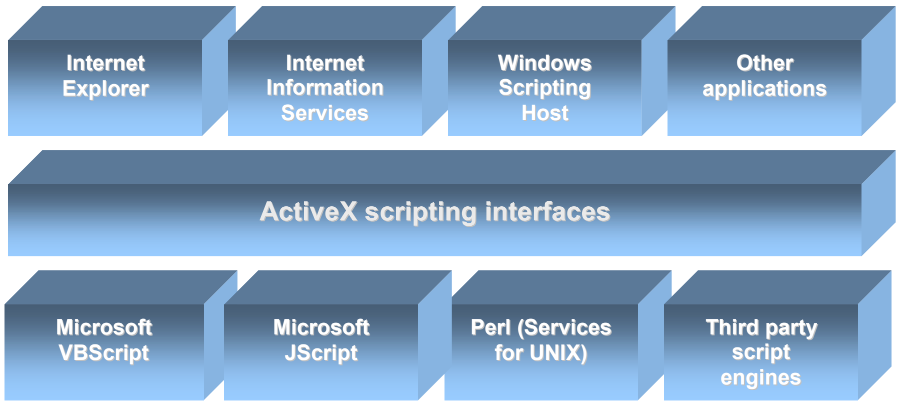


Figure 1 Microsoft Scripting architecture

Using Windows Script Host is not very difficult. If you know Visual Basic, it is easy to transform many simple Visual Basic programs into WSH script files.

Windows Script Host has only one data type called a Variant. This means that the behavior of the Variant type will change based on the context. If you use it with a text string, the variant will behave like a string. If you use it with numbers, the variant will behave like a number.

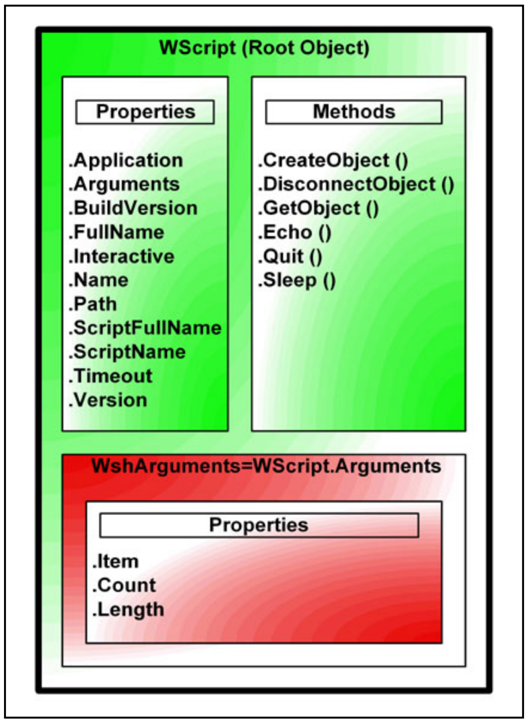
The first Windows Script Host version (officially named Windows Scripting Host 1.0) is usable but has some major restrictions. For instance, it is impossible to use an <include> statement inside a script to reuse existing function from an external file. Under Windows Scripting Host v1.0, all of the code must be physically in a single script file.

With the arrival of the next Windows Script Host version (officially named Windows Script Host 2.0) in July 1999, some of the limitations of v1.0 were removed. Microsoft includes Windows Script Host 2.0 in the release of Windows 2000. Other platforms, such as Windows 9x and Windows NT, will support this version as well. The WSH module can be downloaded from the Web (See Appendix B on page 58).

The object model

Windows Script Host is a COM component provider. The architecture is based on two main COM modules:

Figure 2 The Wscript object



Wscript.Exe: Implements the Wscript object and the WshArguments object. This object is the main component used by any WSH script. It provides all the basic functions used inside a WSH script.
Wshom.Ocx: This is an ActiveX control offering all other WSH objects. (WshShell, WshNetwork, WshShortcut, WshUrlShortcut, WshCollection, WshEnvironment, WshSpecialFolders)

Inside this object collection, only three objects are publicly available. (Objects publicly available have a Prog ID defined in the Windows System.) All other objects, even if they are defined in the Wsh Object model (Wshom.Ocx) must be accessed through one of the three publicly available objects: *Wscript*, *Wscript.Shell* and *Wscript.Network*.

Note: The Prog ID is an identifier of the COM object. To instantiate an object in an application, use the Prog ID as a reference to the desired COM object. Prog IDs are published in the registry HKCR\CLSID.

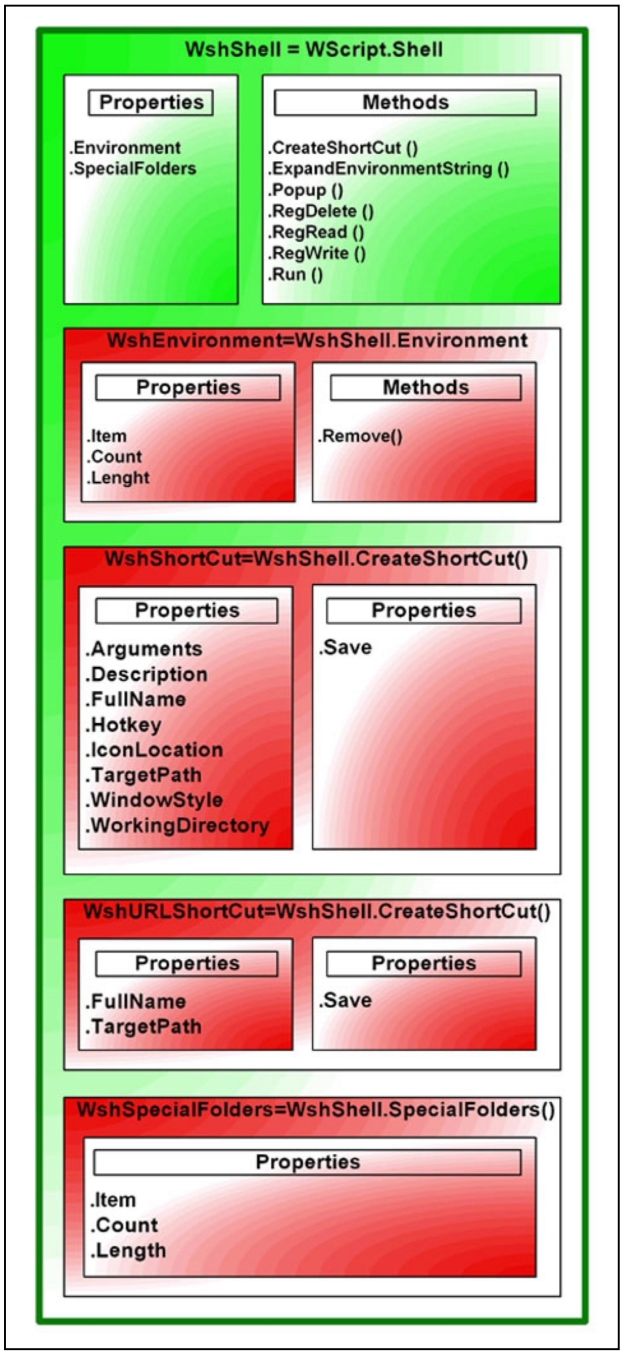
In Figure 2, the *Wscript* object is publicly (in green) visible from the Windows Script Host run-time Wscript.Exe. The *Wscript* object is part of the WSH run-time. So, it is not necessary to instantiate this object. It is available by default in the WSH run-time environment.

On the other hand, the *WshArguments* object can only be accessed from the *Wscript* object itself. This object is not publicly available (in red) because no Prod ID is defined in the Windows system to access it directly. The only way to access the *WshArguments* object is to refer to it via the *Wscript* object as:

```
WshArguments=CreateObject (Wscript.Arguments)
```

Note: A shortened version is used in the rest of the document to ease the presentation in the figures: WshArguments=Wscript.Arguments

Figure 3 The Wscript.Shell object



The *WshArguments* object makes it possible to access arguments passed from the command line to the script. Under WSH 2.0, this object also supports the drag and drop function. From the Windows Explorer, when a file is dropped on the script name, the dropped file name is passed to the script as a regular command line parameter.

As shown in Figures 2, 3, and 4, each of the publicly accessible WSH objects exposes properties and methods as any usual COM object.

Note: The properties and methods listed in Figures 2, 3, and 4 are given as examples only; please refer to the Windows Script Host command reference available from Microsoft for complete properties and methods reference (See APPENDIX B on page 58)

The *WshShell* script object is publicly (in green) available via the Prog ID *Wscript.Shell*. This object contains other objects with their properties and methods and they must be accessed via the *WshShell* object. In this case, objects not publicly available (in red) are:

- *WshEnvironment*,accessed via *WshShell.Environment*.
- *WshShortCut*, accessed via *WshShell.CreateShortcut*.
- *WshURLShortCut*, accessed via *WshShell.CreateShortcut*.
- *WshSpecialFolder,* accessed via *WshShell.SpecialFolder*.

Each object is related to a particular function.

WshShell allows you to manipulate the registry. With this object, a script is able to read, write and delete keys and values from the registry. *WshShell* also gives the ability to run an application from within the script.

WshEnvironment gives access to four variable environments within Windows 2000 (Volatile, Process, System and User). This object, with its associated methods, allows reading, creating and deleting variable values from the environment.

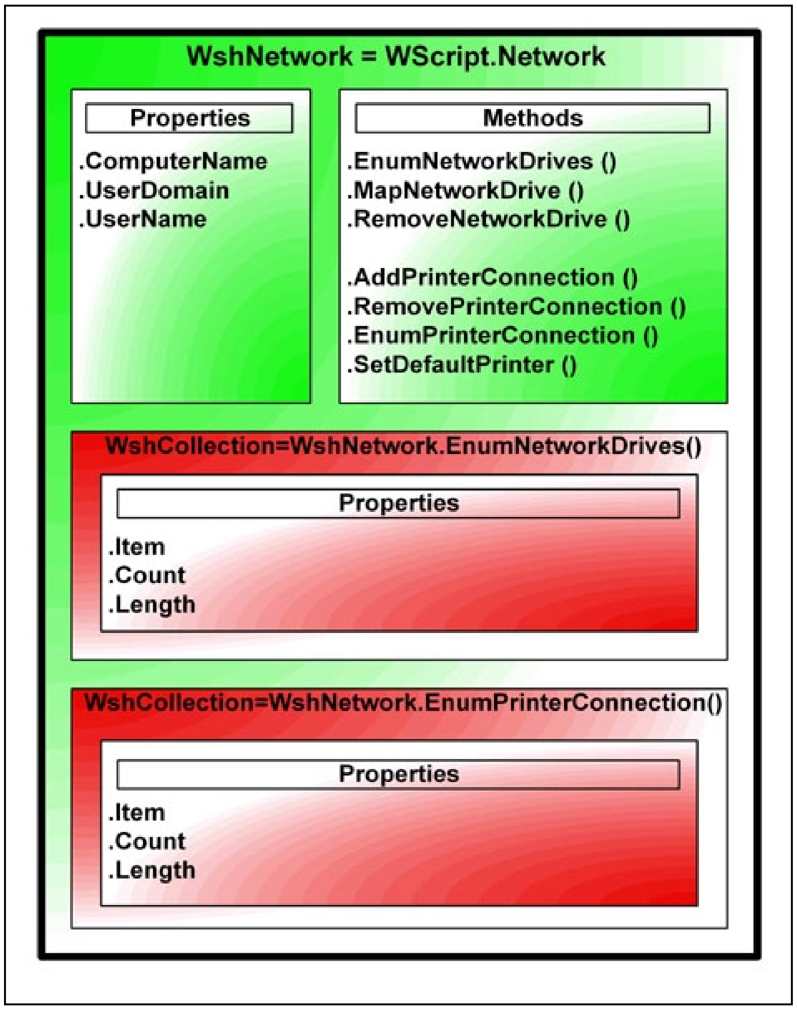
WshShortcut gives access to shortcut characteristics for shortcut creation. All shortcut properties are represented.

WshURLShortcut, similar to the *WshShortcut* object, creates a shortcut containing an URL. Only the extension of the shortcut makes the difference between a traditional shortcut and an URL shortcut.

WshSpecialFolder gives access to the special folders existing on the file system.

The meaning of the special folders are described by their names:

AllUsersDesktop	Fonts	SendTo
AllUsersStartMenu	MyDocuments	StartMenu
AllUsersPrograms	NetHood	Startup
AllUsersStartup	PrintHood	Templates
Desktop	Programs	
Favorites	Recent	



In the same way as the WshShell object, the *WshNetwork* object is publicly (in green) available via the Prog ID:

Wscript.Network

WshNetwork offers access to another object type specifically related to enumerations. Enumerations refer to the *WshCollection* object and are used to enumerate network drive connections and printer connections. These two *WshNetwork* object methods allow access to the *WshCollection* object. The way to access it is by referring to the *WshNetwork* object:

WshCollection=WshNetwork.Enum ...

Figure 4 The *WshNetwork* object

The Windows Script Host Capabilities

Based on the COM object model described in the previous section, you can see that the Windows Script Host supports a variety of uses. Each version of WSH provides different levels of support. Windows Scripting Host 1.0 provides the following features:

- Reading arguments on the command line
- Providing information on the running script itself
- Manipulating the variable environment
- Accessing network information (i.e. such as the current user)
- Creating, retrieving and deleting network drive mappings
- Running external programs
- Manipulating Desktop objects (i.e. creating shortcuts)
- Accessing the registry
- Using registered COM object methods and properties, such as MAPI, ADSI and WBEM (all registered COM providers written for an automation language are usable from within WSH)
- Accessing the file system (via the *Scripting.FileSystemObject* object)

Windows Script Host 2.0 provides all the features supported by version 1.0 and adds new features such as:

- Support for include files:
  It is possible to write a function, put it in a separate file and make reference to that file when you need to use the function.
- Support for multiple engines:
  To permit a better reusability, you can mix languages inside the same script. So, it will be possible to reuse a function written in JavaScript from a script developed in VBScript.
- Support for pausing a script.
- Support for Std In and Sdt Out pipe redirection.
- Support for Drag and Drop:
  A script can be started simply by dragging and dropping a filename on the script name visible in the Windows Explorer.

To get an overview of all new features provided by WSH version 2.0, look at: http://msdn.microsoft.com/workshop/languages/clinic/scripting061499.asp

Using Basic WSH functions in a Windows 2000 Logon Script

All of the code presented in the following pages comes from a Logon Script sample included in the Scripts Kit that accompanies this White Paper. Each basic WSH function is easy to reuse because it is encapsulated in a callable VB script function.

Note: For more information about WSH reusable code, see the second part of this study called “The powerful combination of WSH and ADSI under Windows 2000”.

Functions presented use some global variables declared in the beginning of the script. Be sure to check the complete source code to know which variables are used internally in the functions.

The purpose of the sample codefragments is to show how to use WSH to get specific information, not to show the complete Logon Script logic. Code has been removed in order to highlight the essential WSH operations. Missing lines are represented by an ellipsis (“…:”).

Note: Most of the time, the missing code concerns error handling or the file logging tracing calls This is why the reader will often see an *objFileName* parameter in all presented functions.

Providing information on the script environment and the Windows System

After the usual variable and constant declarations, the first operation is to instantiate the *Wscript.Shell* and the *Wscript.Network* objects (lines 111 and 113 in Sample 1). It is important to keep in mind that the only object available by default under WSH is the *Wscript* object. The three basic objects give access to the whole set of information available about the WSH run-time environment.

Sample 1 Providing information on the running script and Windows System.

```
...:
...:
...:
107:' --------------------------------------------------------------------------------------------------
108:' Create WSH base objects.
109:
110:' For the Shell operations -------------------------------------------------------------------------
111:Set WShell = Wscript.CreateObject("Wscript.Shell")
112:' For the Network operations -----------------------------------------------------------------------
113:Set WNetwork = Wscript.CreateObject("Wscript.Network")
114:
115:' --------------------------------------------------------------------------------------------------
116:WShell.LogEvent 0, "Logon Script started."
...:
128:' --------------------------------------------------------------------------------------------------
129:' Get misc. information for script run-time environment.
130:
131:' Create variable commonly used in the logon script -----------------------------------------------
132:' Information about misc. directories
133:strStartMenu = WShell.SpecialFolders("StartMenu") & "\"
...:
136:strSystemRoot = ReadEnvironmentVariable (objLogFileName, "Process", "SystemRoot") & "\"
137:strUserTemp = ReadEnvironmentVariable (objLogFileName, "User", "Temp") & "\"
...:
139:strSystem32 = strSystemRoot & "system32\"
...:
141:strAllUsersStartMenu = WShell.SpecialFolders("AllUsersStartMenu") & "\"
...:
144:
```

```
145:' Information about Network -----------------------------------------------------------------------
146:strDomainName = WNetwork.UserDomain
147:strUserName = WNetwork.UserName
148:strLocalComputerName = WNetwork.ComputerName
149:strLogonServerName = Mid (ReadEnvironmentVariable (objLogFileName, "Process", "LogonServer"), 3)
150:
151:' Information about script environment -------------------------------------------------------------
152:strScriptName = Wscript.ScriptName
153:strScriptFullName = Wscript.ScriptFullName
...:
155:strScriptingName = Wscript.Name
156:strEngineFullName = Wscript.FullName
157:strEnginePath = Wscript.Path
158:strEngineVersion = Wscript.Version
...:
...:
...:
```

Once WSH basic objects are instantiated and made available, the script can assign miscellaneous information to script variables for use in later processing. For instance, it is possible to know the File System path of the “current user’s Startup folder” (Line 133) or the “All Users Startup Folder” (line 141). Note the appended backslash to the path to ease any future reuse.

Environment variables can be read by using a dedicated function (lines 136, 137 and 149). Instead of using the in-line coding of the WSH methods to read environment variables, this operation is encapsulated in a function. This encapsulation allows bind the function to other useful operations for a Logon Script, such as logging events to a trace file (See Sample 8 on page 24) or error handling.

In a traditional NT Command file (.CMD), the way to identify the current user and his domain membership is to read the environment variables %Username% and %UserDomain%. To determine the computer name, it is possible to read the environment variable %ComputerName%. WSH offers another way to determine this information within a script (line 146 to 148) by using the *Wnetwork* object (object instantiated at line 113).

Information about the script itself is also available through several properties of the *WScript* object (lines 152 to 158).

WSH is able to write a trace in the Windows 2000 application event log by using the *Wshell.LogEvent* method (line 116) to notify the logon script startup.

Reading arguments from the command line

As with other scripts, command line arguments can be specified to WSH scripts. In the case of the Logon Script sample, two arguments are accepted: *Verbose* (To log all the activity in a file) and *ErrorPopup* (To popup any error during the Logon script execution). The sample below shows two pieces of code:

- The first part (lines 163 to 182) verifies if the retrieved arguments are valid arguments accepted by the script.
- The second part of the code (lines 806 to 833) gets the arguments typed on the command line. This second part uses the WSH arguments object to read the command line. As input, the function gets an object file pointer to log trace the operations (See Sample 8 on page 24) and an array pointer to return the arguments list.

Sample 2 Reading arguments from the command line

```
...:
...:
...:
163:' -------------------------------------------------------------------------------------------------
164:' Command line argument reading. ('verbose'or nothing)
165:Dim intIndice
166:Dim strParameterList()
167:
168:If (ReadCommandLineArgument (objLogFileName, strParameterList)) Then
169:   For intIndice = 0 to Ubound(strParameterList) - 1
170:       Select Case Ucase (strParameterList (intIndice))
171:           Case "VERBOSE"
172:                boolVerbose = True
173:                WShell.LogEvent 0, "'Verbose' mode enabled."
174:           Case "ERRORPOPUP"
175:                boolErrorPopup = True
176:                WShell.LogEvent 0, "'ErrorPopup' mode enabled."
177:           Case Else
178:                WShell.LogEvent 1, "Invalid command line parameter detected: '" & _
179:                                   strParameterList (intIndice) & "'."
180:       End Select
181:   Next
182:End If
...:
...:
...:
806:' -------------------------------------------------------------------------------------------------
807:Private Function ReadCommandLineArgument (objFileName, strParameterList)
808:
809:Dim objArguments
810:Dim intIndice
...:
816:        Set objArguments = Wscript.Arguments
817:
818:        If objArguments.Count = 0 Then
819:           intRC = WriteToFile (objFileName, "No command line arguments given")
820:        Else
821:           ReDim strParameterList (objArguments.Count)
822:           For intIndice = 0 To objArguments.Count - 1
...:
825:               strParameterList (intIndice) = objArguments (intIndice)
826:           Next
827:        End If
828:
829:        ReadCommandLineArgument = objArguments.Count
830:
831:        Set objArguments = Nothing
832:
833:End Function
...:
...:
...:
```

The arguments are read by a new object: *Wscript.Arguments* (line 816). Command line parameters are stored in this object and are accessible via an array index. A simple “For … Next” loop allows you to extract each command line argument separately (line 822 to 826). The argument separator is the space character. If an argument contains spaces, it must be enclosed by quotation marks. By using the *Count* property, the script initializes the exact size needed for the array used to return the command line parameters list (line 821)

Manipulating the environment variables

There are four types of environment variables: *Process*, *User*, *System* and *Volatile*. Each environment type contains a specific set of variables. WSH offers methods to read, create and delete environment variables. Each operation is encapsulated in a specific function for the Logon Script. Note that regular Domain users do not have access to the *System* environment. The access to the *System* environment requires administrative privileges.

Sample 3 Manipulating the environment variables

```
...
...
...
strSystemRoot = ReadEnvironmentVariable (objLogFileName, "Process", "SystemRoot") & "\"
strUserTemp = ReadEnvironmentVariable (objLogFileName, "User", "Temp") & "\"
strSystemTemp = ReadEnvironmentVariable (objLogFileName, "System", "Temp") & "\"
...
' -------------------------------------------------------------------------------------------
' Create three new environment variables
intRC = CreateEnvironmentVariable (objLogFileName, "Process", "USERNAMEADsPath", strUserNameADsPath)
intRC = CreateEnvironmentVariable (objLogFileName, "Process", "USERNAMEDN", strUserNameDN)
intRC = CreateEnvironmentVariable (objLogFileName, "Volatile", "SITENAME", strLogonSiteName)
...
' -------------------------------------------------------------------------------------------
Private Function ReadEnvironmentVariable (objFileName, strEnvironmentType, strVarName)

Dim objEnvironment
...
        ' Create a new variable via environment object.
        Set objEnvironment = WShell.Environment (strEnvironmentType)
...
        ReadEnvironmentVariable = WShell.ExpandEnvironmentStrings (objEnvironment(strVarName))
...
        Set objEnvironment = Nothing

End Function

' -------------------------------------------------------------------------------------------
Private Function CreateEnvironmentVariable (objFileName, strEnvironmentType, strVarName, varValue)

Dim objEnvironment
...
        ' Create a new variable via environment object.
        Set objEnvironment = WShell.Environment (strEnvironmentType)
...
        objEnvironment (strVarName)=varValue
...
        Set objEnvironment = Nothing

End Function

' -------------------------------------------------------------------------------------------
Private Function RemoveEnvironmentVariable (objFileName, strEnvironmentType, strVarName)

Dim objEnvironment
...
        ' Remove variable via environment object.
        Set objEnvironment = WShell.Environment (strEnvironmentType)
...
        objEnvironment.Remove (strVarName)

        Set objEnvironment = Nothing
```

```
777:
778:End Function
...:
...:
...:
```

The initial operation is the instantiation of the *Wshell.Environment* object (lines 707, 736 and 766) to a specific environment type. The environment variable to be accessed determines the environment type to select. For instance, to read the %SystemRoot% variable, the script needs to access the *Process* environment. To read the %LogonServer% variable, the script must read the *Volatile* environment.

A particular case of reading an environment variable is when an environment variable contains in its assignment another environment variable name. This is the case for the %Path% variable. The %SystemRoot% variable is part of the %Path% value. To resolve the content of the %Path% variable, the script must invoke the *ExpandEnvironmentString* method (line 713).

Creating, retrieving and deleting network connections

The network operations can be completed via the *Wnetwork* object. With this object, a script is able to complete network drive or printer connections. The method is the same, only the parameters are different if the script connects to a drive or to a printer.

Note: Under ADSI, much more network information can be retrieved; see the code examples beginning with Sample 14 on page 50 for additional examples.

Some basic network information (such as the domain and username of the user currently logged in, local machine name and logon server) can be retrieved as shown in Sample 1 on page 15 (line 146 to 149).

Sample 4 Creating, retrieving and deleting network connections

```
...:
...:
...:
271:   intRC = EnumerateDriveConnections (objLogFileName)
272:   intRC = ConnectNetworkDrive (objLogFileName, "V:", cHomeDirectoryRootShare)
273:   intRC = EnumerateDriveConnections (objLogFileName)
274:   intRC = DisconnectNetworkDrive (objLogFileName, "V:", True)
...:
509:' -------------------------------------------------------------------------------------------
510:Private Function EnumerateDriveConnections (objFileName)
511:
512:Dim enumDrives
513:Dim intIndice
...:
519:        Set enumDrives = WNetwork.EnumNetworkDrives
...:
526:        If enumDrives.Count = 0 Then
527:           intRC = WriteToFile (objFileName, "No drive to list.")
528:        Else
529:           intRC = WriteToFile (objFileName, "Current network drive connections:")
530:           For intIndice = 0 To enumDrives.Count - 1 Step 2
531:               intRC = WriteToFile (objFileName, enumDrives (intIndice) & " -> " & _
532:                                                  enumDrives (intIndice + 1))
533:           Next
534:        End If
535:
536:End Function
537:
```

```
538:' -----------------------------------------------------------------------------------------------
539:Private Function ConnectNetworkDrive (objFileName, strDriveLetter, strShareName)
...:
547:        WNetwork.MapNetworkDrive strDriveLetter, strShareName
...:
554:End Function
555:
556:' -----------------------------------------------------------------------------------------------
557:Private Function DisconnectNetworkDrive (objFileName, strDriveLetter, boolConfirm)
558:
559:Dim intClick
560:
561:        On Error Resume Next
562:
563:        If boolConfirm Then
564:           intClick = WShell.Popup ("Remove Network Drive connection '" & _
565:                                    UCase (strDriveLetter) & "' (Y/N) ?", _
566:                                    0, _
567:                                    "(LogonScript) Confirmation", _
568:                                    cQuestionMarkIcon + cYesNoButton)
569:           If intClick = cNoClick Then
570:              Exit Function
571:           End If
572:        End If
...:
577:        WNetwork.RemoveNetworkDrive strDriveLetter
...:
583:
584:End Function
...:
...:
...:
```

The script above shows how to enumerate the current network drive connections before creating a new drive connection (line 271, 272). The drive connection enumeration uses a specific object for this purpose (line 519). Once the *enumeration* object is created, a “For … Next” loop can retrieve all existing connections (lines 530 to 533). The existence of connections is tested by the *count* property available from the *enumeration* object (line 526). The connections are available as an array from the *enumeration* object. Even array elements contain the drive letter used for the connection; odd array elements contain the Universal Naming Convention (UNC) mapping used (lines 531, 532).

Note: The display output of the enumeration is written to a file by invoking the “WriteToFile” function. See Sample 8 on page 24 for more details about the “WriteToFile” function.

The connection and the disconnection methods are pretty easy to use. The connection needs the drive letter and the UNC (line 547). The disconnection needs the drive letter only (line 577). To complete the disconnection process, a supplemental parameter (line 557, parameter boolConfirm) is added to the function input. This parameter specifies whether the user needs to confirm the drive disconnection (line 563 to 572). The user popup function is done via the *Wshell* object popup method (line 564).

Network printer connections use the same methods and logic. See the Logon Script sample provided in the Sample Scripts Kit accompanying this White Paper for the complete sample functions.

Running external programs

This functionality is not used in the current Logon Script sample, but some circumstances may require use of external applications. To start an application from a script, the *Wshell* object provides the *run* method. Only one line is needed:

WshShell.run "Notepad.Exe", intWindowStyle, boolWait

The “intWindowStyle” parameter determines the window behavior of the started application. Based on that value, the window will be hidden (0), normal active (1 or 5), minimized active (2), maximized (3), visible but not activated (4) or minimized (6). These values can be assigned to meaningful constants in the beginning of the script to make the code easier to read.
Note: One of these constants is used for the shortcut creation in Sample 5.

The “boolWait” parameter determines if the method has to wait until the started application is closed to continue the script execution.

Creating shortcuts

WSH provides methods to create shortcuts in any folder available on the File System. Basically, shortcuts can be created in any *SpecialFolders* such as the Desktop folder (See page 13 for the *SpecialFolders* list). Shortcut creation is available directly from the *Wshell* object. The procedure is relatively explicit. All parameters available from a traditional shortcut GUI can be referenced through the *objShortcut* object in Sample 5 below.

Sample 5 Manipulating Desktop objects

```
...
...
...
' ----------------------------------------------------------------------------------------------------
' Create a Shortcut on the Desktop for this particular user.
Dim objShortCut

intRC = WriteToFile (objLogFileName, "** Create a shortcut on the Desktop.")

Set objShortCut = WShell.CreateShortCut(strDesktop & "\Windows Getting Started.Lnk")
objShortCut.Description = "Getting started with Windows NT Server"
objShortCut.Arguments = ""
objShortCut.HotKey = "ALT+CTRL+S"
objShortCut.IconLocation = strSystem32 & "\wizmgr.exe"
objShortCut.TargetPath = strSystem32 & "\wizmgr.exe"
objShortCut.WindowStyle = SW_SHOWNORMAL
objShortCut.WorkingDirectory = strSystem32
objShortCut.Save
...
...
...
```

Accessing the registry

Access to the registry is available through the *Wshell* object. This object exposes three methods to read, create and delete registry keys or registry key values. If the user security context has the required permissions, any registry key can be accessed.

Sample 6 Accessing the registry

```
...
...
...
strNTVersion = ReadRegistry (objLogFileName, _
                             "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion", _
                             "CurrentVersion", _
                             "REG_SZ")
...
```

```
' ---------------------------------------------------------------------------------------------
...
' Creating some "dummy" registry keys to show how to do.
intRC = WriteRegistry (objLogFileName, "HKCU\Software\Compaq\Registry Access", _
                                       "ValueRegBinary", "REG_BINARY", 65534)
...
Dim varRegValue

varRegValue = ReadRegistry (objLogFileName, "HKCU\Software\Compaq\Registry Access", _
                                            "ValueRegBinary", "REG_BINARY")
...
' ---------------------------------------------------------------------------------------------
' Delete the registry (Be sure that user has right to create it)
...
intRC = DeleteRegistry (objLogFileName, "HKCU\Software\Compaq\Registry Access", _
                                        "ValueRegBinary")
...
' ---------------------------------------------------------------------------------------------
Private Function ReadRegistry (objFileName, strKeyName, KeyValueName, strRegType)

Dim strRegKey
Dim varRegKeyValue

Dim strTempValue
Dim strChar
Dim strTemp
...
        strTempValue = WShell.RegRead (strRegKey)
...
        Select Case strRegType
               Case "REG_BINARY"
                    For Each strChar In strTempValue
                        strTemp = Right("00" & hex(strChar), 2) & strTemp
                    Next
                    varRegKeyValue = "0x" & strTemp
               Case "REG_DWORD"
                    varRegKeyValue = "0x" & Hex (strTempValue)
               Case Else
                    varRegKeyValue = strTempValue
        End Select
...
        ReadRegistry = varRegKeyValue

End Function

' ---------------------------------------------------------------------------------------------
Private Function WriteRegistry (objFileName, strKeyName, KeyValueName, strRegType, varRegKeyValue)

Dim strRegKey
...
        strRegKey = strKeyName & "\" & KeyValueName
...
        WShell.RegWrite strRegKey, varRegKeyValue, strRegType
...
End Function

' ---------------------------------------------------------------------------------------------
Private Function DeleteRegistry (objFileName, strKeyName, KeyValueName)

Dim strRegKey
...
        strRegKey = strKeyName & "\" & KeyValueName
...
        WShell.RegDelete strRegKey
```

```
 ...:
 477:        intRC = WriteToFile (objFileName, strRegKey & " -> DELETED")
 478:
 479:End Function
 ...:
 ...:
 ...:
```

The general rule for accessing the registry is to provide the following parameters:

- The Key Name
- The Key Value Name
- The Value for the given Key Name

All methods to manipulate keys have the same behavior. If the passed parameter ends with a backslash '\', then the invoked method addresses a key path. If the passed parameter does not end with a backslash '\', then the invoked method addresses a key value name. This is why the sample functions concatenate the key path and the key value name (line 408, 440 and 468). If the key value name is empty, then the parameter passed to the WSH registry method ends with a backslash and will address a key name. In this way, a single function can be used for both key names and key value names.

The only complexity in the sample code is the handling of binary values. A binary registry key value needs to be converted to a hexadecimal string. This is the purpose of lines 416 to 419.

Using registered COM objects

Any COM object publicly available from the Windows system (and written for automation language usage) can be instantiated from WSH. For instance, Message Application Programming Interface (MAPI), Active Directory Service Interfaces (ADSI) and many other objects can be referenced from a WSH script. For example, the Logon Script sample contains a function that shows the user the number of unread mail in his mailbox. This function uses MAPI. MAPI is out of the scope of this White Paper, but it is a nice feature to have in a Logon Script. Other samples using ADSI objects are provided on page 50 and following pages.

Sample 7 Using registered COM objects

```
 ...:
 ...:
 ...:
 979:' -----------------------------------------------------------------------------------------------
 980:Private Function CheckMAPIMail (objFileName, strMAPIProfileName, strUserName, boolPrompt)
 981:
 982:Dim MAPISession
 983:Dim MAPIMessages
 ...:
 996:        Set MAPISession = Wscript.CreateObject ("MSMAPI.MAPISession")
 997:        Set MAPIMessages = Wscript.CreateObject ("MSMAPI.MAPIMessages")
 ...:
1005:        MAPISession.DownLoadMail = False
1006:
1007:        ' Do not prompt the user for the MAPI profile, use the default
1008:        MAPISession.LogonUI = False
1009:        MAPISession.UserName = strMAPIProfileName
1010:
1011:        ' Signon method.
1012:        MAPISession.SignOn
....:
1018:        MAPISession.NewSession = True
```

```

        ' Associate the MAPIMessages session to current opened MAPI session
        MAPIMessages.SessionID = MAPISession.SessionID

        MAPIMessages.FetchUnreadOnly = True
        MAPIMessages.Fetch

        ' Test the 'MAPIMessages.MsgCount' and skip prompting if equal to zero
        ' If counter is equal to zero, its needless to show this information.
        CheckMAPIMail = MAPIMessages.MsgCount
....:
        If MAPIMessages.MsgCount And boolPrompt Then
           WShell.Popup "Today, you have " & MAPIMessages.MsgCount & " unread mail(s).", _
                        0, _
                        "(LogonScript) Hello, " & strUserName, _
                        cInformationMarkIcon Or cOkButton
        End If

        ' Close the session.
        MAPISession.SignOff

        ' Flag for new session.
        MAPISession.NewSession = False
....:

End Function
....:
....:
....:
```

Accessing the file system

The File System can be accessed from WSH via a specific object. It is important to understand that the File System object is not an object provided by WSH. As MAPI can be used from WSH, the FileSystem object follows the same rules. To access the File System, the script needs to instantiate the File System object by its publicly available ProgID:

Scripting.FileSystemObject

Once created, the *FileSystemObject* offers several methods to manipulate files. A full examination of the FileSystemObject is outside of the scope of this White Paper, but for more information about the object please refer to APPENDIX B on page 58.

The Logon Script uses the FileSystem object to trace all the operations executed during Logon. The purpose is to ease the troubleshooting of the logon process. Only three basic operations are used: file creation, writing data to the open file and closing the file.

The trace logging function is important because once the Logon Script is published in the Windows 2000 Group Policies; it is executed in the background. Having a trace logging function based on a parameter ('verbose' in Sample 2) will make it easier to locate any potential problems or errors during the script execution.

Sample 8 Accessing the file system

```
...:
' -------------------------------------------------------------------------------------
' Create the Text file for logging.
Set objLogFileName = CreateTextFile (strLogFileName)
```

```
...
' Put a trace of logon script startup in log file.
intRC = WriteToFile (objLogFileName, "** Logon started " & _
                                      FormatDateTime (Date, vbLongDate) & " at " & _
                                      FormatDateTime (Time, vbShortTime) & ".")
...
' -------------------------------------------------------------------------------------------------
intRC = WriteToFile (objLogFileName, "** Closing Log file.")
CloseTextFile (objLogFileName)
...
' -------------------------------------------------------------------------------------------------
Private Function CreateTextFile (strFileName)

Dim objFileName

        On Error Resume Next

        Set objFileSystem = Wscript.CreateObject ("Scripting.FileSystemObject")
....
        Set objFileName = objFileSystem.CreateTextFile (strFileName, True)
....
        Set CreateTextFile = objFileName

End Function

' -------------------------------------------------------------------------------------------------
Private Function CloseTextFile (objFileName)

        If boolVerbose Then
           objFileName.Close

           Set objFileName = Nothing

           Wscript.DisconnectObject objFileSystem
           Set objFileSystem = Nothing
        End If

End Function

' -------------------------------------------------------------------------------------------------
Private Function WriteToFile (objFileName, Text)

        If boolVerbose Then
           If Mid(Text, 1, 3) = "** " Then
              objFileName.WriteLine (Text)
           Else
              objFileName.WriteLine "   " & Text
           End If
        End If

End Function
```

Support for include files

Including external scripts in a main script enables code reusability. With WSH version 1.0, it was mandatory to have one single file containing all the code with all the functions in it. A second script was not able to reuse functions from that single file without copying and pasting code from one file to the other. This process duplicates pieces of code and creates problems in terms of code maintenance.

Support for file inclusion is implemented in WSH version 2.0 through the use of XML sections. The programmer can have several script files containing miscellaneous functions. By using the XML sections, the programmer can refer to each file to reuse the defined functions.

This feature is examined in more detail in the second part of this white paper called “The powerful combination of WSH and ADSI under Windows 2000”.

Support for multiple engines

Using include files allows the reuse of previously developed functions, but what about the language used to code the included function? WSH version 2.0, based on the same XML sections principle, allows the programmer to specify the language of the included function. In this case, the developer can reuse a function written in JavaScript from a VB script (and vice versa).

This feature is examined in part 2 of this white paper called “The powerful combination of WSH and ADSI under Windows 2000”.

Pausing a script

The ability to pause a script is not used in the current Logon Script sample. However, there are circumstances where such a function can be useful. The WSH method is available from the *Wscript* object and uses only one parameter: the number of milliseconds to pause the script.

Wscript.Sleep (MillSeconds)

Drag and Drop support

The drag and drop support is implemented in WSH version 2.0 only. This support enables the user to start a script by dragging and dropping a file on the desired script name. From the programming point of view, this is transparent because the dropped file name is presented to the script as an argument from the command line. So, to capture a dropped file name, the script must use the logic of Sample 2 on page 17.

Standard Input and Standard Output

WSH version 2.0 provides support for standard input and output pipes. This new feature lets you pipe input and output between scripts or any other applications from the command line. For instance, if you type the following on the command line, output of the DIR command is piped to the script:

DIR | Cscript.Exe StdInOut.vbs

Sample 9 demonstrates how the output of the DIR command can be written to the standard output. Of course, it was also possible to redirect the output to the standard error (Wscript.StdErr).

Sample 9 Standard Input and Standard Output usage

```
' VB Script demonstrating a Standard Input and Standard Output usage                '
'                                                                                   '
' Version 1.00 - Alain Lissoir                                                      '
' Compaq Computer Corporation - Professional Services - Belgium -                   '
'                                                                                   '
' Any comments or questions:                       EMail:alain.lissoir@compaq.com '

Option Explicit

Dim strInputString
Dim stdOut, stdIn

```

```
Set stdOut = WScript.StdOut
Set stdIn  = WScript.StdIn

Do While Not stdIn.AtEndOfStream

   strInputString = stdIn.ReadLine
   stdOut.WriteLine "String piped: '" & strInputString & "'"

Loop
```

ACTIVE DIRECTORY SERVICE INTERFACES

Description

ADSI stands for Active Directory Service Interfaces. ADSI is the primary interface to the Windows 2000 Active Directory. More than Windows 2000, ADSI provides access to many other data sources. Some of them are:

- Windows NT 4.0 SAM Database
- Internet Information Server 4.0
- Site Server 3.0
- Exchange Server 5.5
- Netware 3x, 4.x and 5.0.

Using ADSI represents a direct benefit for administrators. ADSI gives easy access to objects in the directory and OS. Scripts and programs can access these directory objects through COM interfaces. Moreover, COM is the easiest way for scripts to communicate with some Operating System components. Whatever the scripting language used (JavaScript, VBScript or Perl), the way in which the COM object is used is independent. It is independent because the COM object itself exposes available methods and properties. Of course, the language engine must be COM-aware to be able to communicate with the object.

For instance, you can use ADSI from an ASP page to read or write specific data in an Exchange Directory, but you can also use ADSI from Perl, JavaScript, or VBScript to create users in a NT 4.0 domain, or in an Active Directory Domain.

Each time an object in the Active Directory is accessed, a COM object method is used. Each time some object properties are read (or written) from the Active Directory, COM methods and a set of properties are used.

Combining the facility of Windows Script Host and ADSI provides the tool needed for powerful logon scripts, unattended setup and remote administration.

Before ADSI, when an administrator (or developer) wanted access to some directory information, he had to use a specific SDK. The SDK used depends on the platform accessed. Now, with this unified way of programming, you can access Directory Service objects in a heterogeneous environment.

The main purpose of ADSI is to abstract the directory service. Even if a Directory Service can have a specific name space, most ADSI objects are namespace independent (See APPENDIX A on page 56)

To bind to a data source, choose a name space provider. Several name space providers exist and they each provide access to different data sources. Here are the available name spaces:

- **LDAP:** The LDAP name space provides access to the Windows 2000 Active Directory, the Exchange Server Directory, a Netscape server Directory, or any other LDAP host
- **WinNT:** The WinNT name space provides access to the Windows NT 4.0 SAM database or to a Windows 2000 Active Directory server seen as a NT 4.0 legacy system.

- **IIS:** The IIS name space provides access to the Internet Information Server 4.0 configuration and settings. For instance, you can create a virtual directory configuration from a script by using ADSI.
- **NWCOMPAT:** The NWCOMPAT name space provides access to a Netware server in bindery mode (native NetWare server mode such as Novell 3.x servers).
- **NDS:** The NDS name space provides access to the Netware Directory Service available on Netware 4.x and 5.x servers. Note that Novell also has a LDAP access to its Directory service.

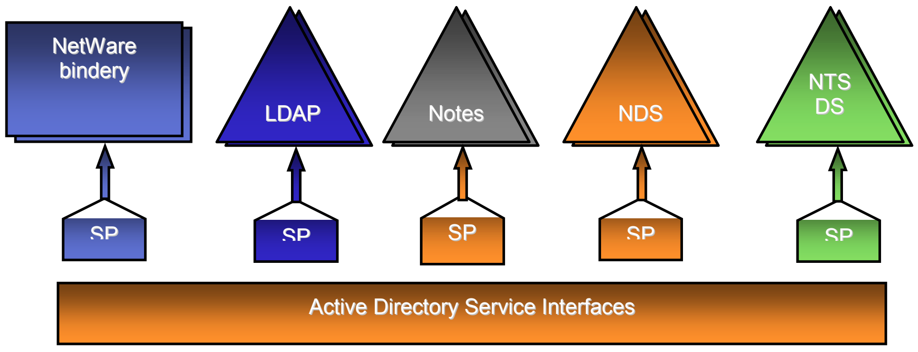


Figure 5 ADSI Service Providers

A service provider has to be used to access a name space. The service provider translates the ADSI method into a Directory Service specific API (DS-Api) call. At this point, ADSI makes the abstraction of the Directory accessed.

Of course, once a namespace is selected, there is a specific syntax related to that namespace. More than this, the LDAP access has also to take into account the Directory structure to access objects. For instance, the Exchange 5.5 Directory is not organized in the same way as a Windows 2000 Active Directory, even if a script uses the same namespace (LDAP) to access them.

It is very important to understand that ADSI abstracts the way a directory is accessed, but not the directory structure itself.

ADSI implementation

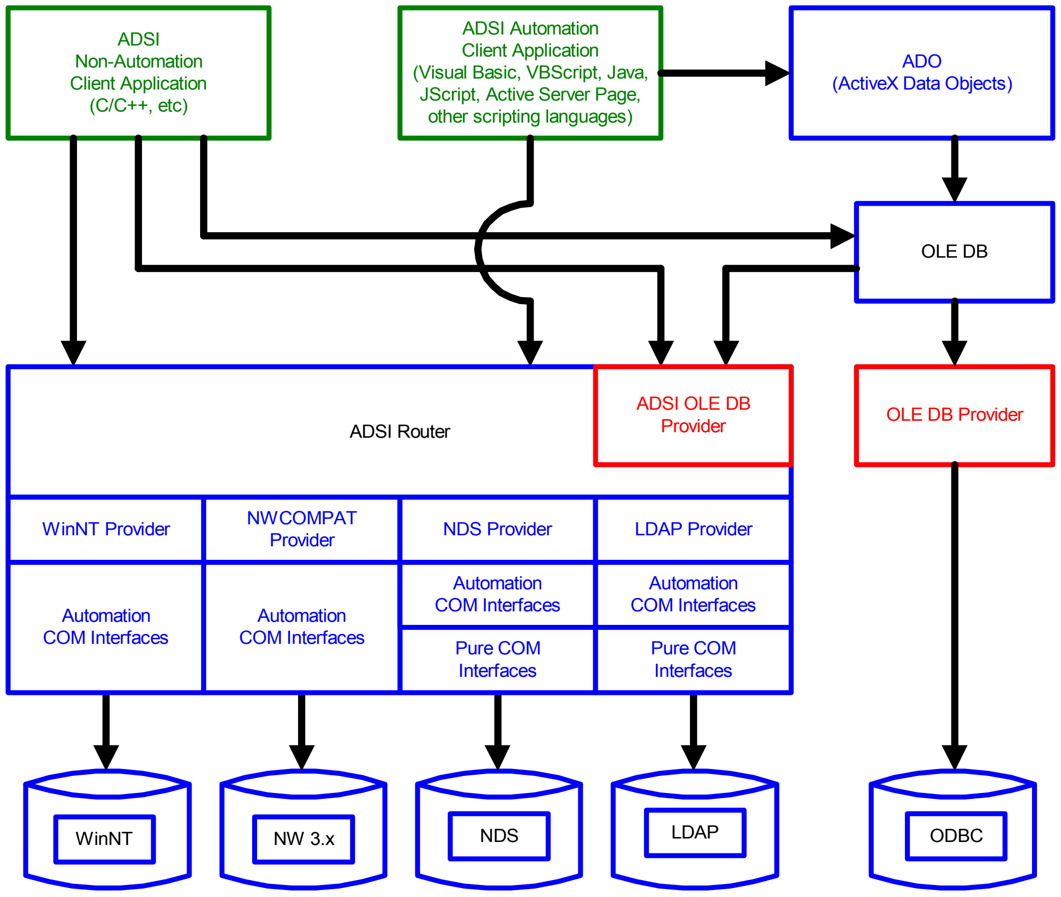


Figure 6 ADSI Architecture and language support.

Whatever the access mechanism, interaction with data sources can be done from many programming languages. The language used (Automation or non-automation) determines the interface type used to complete the task (See Figure 6 above). Note the particular access to the ADSI OLE DB provider via ADO for the automation languages.

Note: Part two of this document named "The powerful combination of WSH and ADSI under Windows 2000" will cover how to use this particular interface to make queries in the Active Directory.

Behind the word "access", it is important to understand that there are several types of operations provided by ADSI, including:

- Bind to a DS object:

  This operation links an application to an object of the Directory. The bind operation can be executed in two forms: using the current credentials or using a specific credential. Using specific credentials allows the script to override the security limitation of a user running the application. In other words, the script can access objects in a different security context from the application.

- Enumerate objects within a DS object:

  Once binding is complete, it is possible to enumerate all the objects contained within the DS object (that is, enumerating all the child objects present in a container).

- Read from and write to properties (attributes) of a DS object:

  Changing a property on an object means that the object must be accessed first. To know which properties are available for an object, it is mandatory to query the schema. The schema will give information about the syntax to be used for each property access.

- Manage the schema of a DS:

  One of the most important purposes of a Directory Service is to be extensible. This is why there is a schema. The schema contains all the object classes, the properties associated to objects, the property syntax and the relation between all of them. Having a schema permits to add, remove, change objects published in the Directory Service. ADSI provides also a way of access to the schema.

- Manage security on DS objects:

  More than managing objects, changing the security on available objects is also possible. This security management can also be executed through some ADSI SDK extensions. There are some restrictions related to the name space used.

- Submit queries to a DS and return result sets:

  ADSI is providing access through a given name space. More than this, ADSI is providing an ADO read-only interface for searching the data source. When an application wants to make perform a search through ADSI, it can connect to the OLE DB interface provided by ADSI. The application can then submit a search in a query form in the same way as an SQL Query (or a RFC LDAP syntax).

ADSI objects and namespaces

ADSI exposes a lot of publicly available objects. Most objects exposed by ADSI are objects commonly used in the Active Directory. For instance, in the Active Directory Service, there is a user object, so ADSI exposes an *IADsUser* interface.

When using an ADSI object, its behavior is determined by the namespace used to address the object. For instance, under the LDAP namespace, a script can access the **lastName** attribute of a user object. Under the WinNT namespace, this attribute is not available because ADSI takes care of what is available in the directory.

Note: To list all of the objects (with methods and properties) that ADSI makes available would necessitate an entire book into itself. Please refer to APPENDIX A on page 56 for a list of available ADSI interfaces in regards of the name space used (or refer to the Microsoft Active Directory Service Interfaces SDK help file for the complete list of supported interfaces and properties. Make a search on "Provider Support of ADSI Interfaces" to locate the tables).

To ease the ADSI understanding, reader should concentrate on four big interfaces: IADsOpenDSObject, IADs, IADsContainer, and Searches via ADO. Almost 90% of the Active Directory tasks can be done using just these interfaces. Two others important ones are IADsUser and IADsGroup.

HOW IS THE ACTIVE DIRECTORY STRUCTURED?

The Active Directory is a LDAP v3.0 accessible directory. Therefore, it is important to know how the Active Directory looks from an LDAP point of view.

Note: This paper does not examine the Active Directory architecture. For more information on this, please refer to the White Paper "Active Directory - A technical overview" by Micky Balladelli.

The RootDSE

Any respectable LDAP v3.0 server has a **RootDSE** object. The purpose of this object is to provide information about the directory service. The information is located in a set of attributes associated with that object.

Under Windows 2000, when an application accesses the Active Directory Domain objects, it first contacts the **RootDSE** to determine what is the default Windows 2000 Domain name. In the same way, if it wants to get information about available object classes in the Active Directory (with their associated attributes and syntaxes), it must access the Schema context. The Schema access path is also published by one of those **RootDSE** attributes.

The **RootDSE** RFC definition can be found at http://www.rfc-editor.org/rfc/rfc2251.txt.

Here is the list of attributes available from a **RootDSE** object under Windows 2000:

currentTime:	Current time set on this directory server.
subschemaSubentry:	Distinguished name for the subSchema object. The subSchema object contains attributes that expose the supported attributes (in the **attributeTypes** attribute) and classes (in the **objectClasses** attribute). The **subschemaSubentry** attribute and subschema are defined in LDAP 3.0 (see RFC 2251).
dsServiceName:	The distinguished name of the NTDS (NT Directory Server) settings object for this directory server. This path can be helpful to determine the links the server has with other partners.
namingContexts:	Multi-valued. Contains a distinguished name for all naming contexts stored on this directory server. By default, a Windows 2000 domain controller contains at least three contexts: Schema, Configuration, and one for the domain of which the server is a member.
defaultNamingContext:	By default, the distinguished name for the domain of which this directory server is a member. Under Exchange 5.5, the value returned will be the organization name (o=OrgName)
schemaNamingContext:	Distinguished name for the schema container.
configurationNamingContext:	Distinguished name for the configuration container.
RootDomainNamingContext:	Distinguished name for the first domain in the Forest that contains the domain of which this directory server is a member.
SupportedControl:	Multi-valued. Object Identifiers (OID) for extension controls supported by this directory server.
SupportedLDAPVersion:	Multi-valued. LDAP versions (specified by major version number) supported by this directory server.

HighestCommittedUSN:	Highest USN used on this directory server. Used by directory replication.
SupportedSASLMechanisms:	Security mechanisms supported for SASL (Simple Authentication and Secutiry Layer) negotiation (see LDAP RFCs).
DnsHostName:	DNS address for this directory server.
LdapServiceName:	Service Principal Name (SPN) for the LDAP server. Used for mutual authentication.
ServerName:	Distinguished name for the server object for this directory server in the configuration container.

Note: By default, Exchange 5.5 Server acts as an LDAP v3.0 server. This means that a **RootDSE** object is accessible. This Exchange's **RootDSE** object does not have all the same attributes as a Windows 2000 Domain Controller. Listed below are the attributes available from an Exchange 5.5 server **RootDSE** object:

- currentTime
- subschemaSubentry
- defaultNamingContext
- SupportedControl
- namingContexts
- HighestCommittedUSN

Sample 10: Looking at some RootDSE object's attributes.

```
' VB Script accessing RootDSE object to show all associated attributes       '
'                                                                            '
' Version 1.00 - Alain Lissoir                                               '
' Compaq Computer Corporation - Professional Services - Belgium -            '
'                                                                            '
' Any comments or questions:                  EMail:alain.lissoir@compaq.com '

Option Explicit

Dim objRoot
Dim strMember

' -------------------------------------------------------------------------------------------------
WScript.Echo

WScript.Echo "Script RootDSE.vbs showing some RootDSE attributes"
WScript.Echo

Set objRoot = GetObject("LDAP://RootDSE")

' If we contact an Exchange server, we do not get an error for the Windows 2000 related attributes.
On Error Resume Next

' These attributes are Windows 2000 related.
WScript.Echo "configurationNamingContext='" & objRoot.Get("configurationNamingContext") & "'"
WScript.Echo "RootDomainNamingContext='" & objRoot.Get("RootDomainNamingContext") & "'"
WScript.Echo "schemaNamingContext='" & objRoot.Get("schemaNamingContext") & "'"

' These attributes are Windows 2000 and Exchange 5.5 related.
WScript.Echo "DefaultNamingContext='" & objRoot.Get("DefaultNamingContext") & "'"
WScript.Echo "currentTime='" & objRoot.Get("currentTime") & "'"
WScript.Echo "subschemaSubentry='" & objRoot.Get("subschemaSubentry") & "'"
WScript.Echo "defaultNamingContext='" & objRoot.Get("defaultNamingContext") & "'"
WScript.Echo "HighestCommittedUSN='" & objRoot.Get("HighestCommittedUSN") & "'"

```

```
36:' We use a loop to examine a multi-valued attribute, so we get all the values
37:For Each strMember In objRoot.Get("SupportedControl")
38:    WScript.Echo "SupportedControl='" & strMember & "'"
39:Next
40:
41:' We use a loop to examine a multi-valued attribute, so we get all the values
42:For Each strMember In objRoot.Get("namingContexts")
43:    WScript.Echo "namingContexts='" & strMember & "'"
44:Next
45:
46:Set objRoot = Nothing
47:
48:WScript.Quit (0)
```

The sample above is getting the **RootDSE** object's attributes (line 19). The next statement block gets **RootDSE** attributes existing only for Windows 2000 (lines 25 to 27). An "On Error Resume Next" statement is set to avoid any problem if this script is run with an Exchange 5.5 server (line 22). The next statement block is getting the attributes that are common to Windows 2000 and Exchange 5.5 (lines 30 to 44). The last two blocks show how to get and display a multi-valued attribute (lines 37 to 44).

Note: See on page 41: "The object syntax differences" for more information about multi-valued attributes and how to retrieve the information they contain.

The Naming Contexts

Once connected to the **RootDSE**, the script will have to access a naming context. The naming context choice depends on the type of information the script must retrieve. Here are some sample questions:

- **Q1:** Do you want to have more information about an object or more about a syntax attribute?
- **Q2:** Do you want to read or change a site configuration in the organization?
- **Q3:** Do you want to read or change a user group membership?

Each of these questions is addressed by a different naming context. By looking at the **RootDSE**, it is possible to know what is the exact path to use for each related naming context.

The three questions address the following naming contexts:

- **SchemaNamingContext** to get knowledge about the object and attributes available with the syntax used by the attributes. (Q1) You are accessing the Directory Schema owned by one of the Flexible Single Master Operations (FSMO) server role and replicated in the enterprise.

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

- **ConfigurationNamingContext** to access sites and subnet definitions. (Q2)
- **DefaultNamingContext** or **RootDomainNamingContext** if the user object is in the same Domain or in the Root Domain. (Q3)

Note: Exchange 5.5, has several naming contexts. The only difference is that the multi-valued attribute called **namingContexts** must be used because the Exchange's **RootDSE** does not expose each available context as a separate attribute. The only context exposed as a separate attribute is the **defaultNamingContext** containing the Exchange organization name.

The Schema Naming Context

In addition to knowing its distinguished name, it is important to know which attributes are available for an object. Most of the ADSI objects implement a COM interface for each available object in the directory. (i.e. *IADsUser*, *IADsComputer*, …) Of course, this will only work for "standard" objects or objects for which sufficient exposed information is available. What about objects not represented by ADSI? What about an object's attributes? We will see how we can use ADSI and the Active Directory Schema to get information about objects, their available attributes and the syntax to be used for each of those attributes.

The key is that everything resides in the Active Directory Schema. The Schema is the part of the Directory containing all the possible object definitions present in the directory. Querying the Schema is fundamental to knowing more about objects. The schema can also help you write code that is independent of the object/ attributes used in a script.

The Schema Naming Context contains all the classes and attributes and is replicated throughout the entire Forest.

The Configuration Naming Context

The Configuration Naming Context contains all the objects that represent the structure of the Active Directory in terms of domains, Domain Controllers, sites, and other configuration type objects. It shows the topology of the forest and is replicated throughout the forest.

All the objects and their attributes included in this context are defined in the Schema Naming Context.

The Domain Naming Context

The Domain Naming Context contains all the data within a domain. It is replicated only within the domain.

All the objects and their attributes included in this context are defined in the Schema Naming Context.

The Default Naming Context

The Default Naming Context is a Domain Naming Context. Each Windows 2000 Domain Controller is attached to a specific Windows 2000 domain. The Default Naming Context contains the name of current domain.

All the objects and their attributes included in this context are defined in the Schema Naming Context.

The Root Naming Context

The Root Naming Context is a Domain Naming Context. It is the name of the Root domain in the forest. Except from that, it looks like a regular Domain Naming Context.

All the objects and their attributes included in this context are defined in the Schema Naming Context.

ACCESSING THE DIRECTORY

By accessing the **RootDSE**, you are accessing the first visible part of the Windows 2000 Active Directory. The following pages describe how to explore the directory.

The Binding operation

The Namespace

A bind operation is always performed in a specific namespace. Whatever the method used (**GetObject** or **OpenDSObject**) a namespace must always be specified.

Note: See below: "Using other credentials" for more details about **GetObject** or **OpenDSObject** methods.

Under Windows 2000, there are three native namespaces:

LDAP: This namespace will give access to the Windows 2000 Active Directory. By default, TCP port 389 will be used as described by the RFC. This namespace will return all the accessed objects in a syntax related to this LDAP directory namespace.

GC: This namespace will give access to the Windows 2000 Global Catalog. TCP port 3268 will be used. This is a Microsoft implementation. The same type of access is possible by using LDAP on port 3268 (Overwriting the default 389). The syntax of this namespace is the same as the LDAP namespace.

WinNT: This namespace will give access to the NT namespace as it is under Windows NT 4.0. This is a Microsoft implementation. Accessing a Windows 2000 system with this namespace makes the Windows 2000 server look like a Windows NT 4.0 server. No object that is specific to Windows 2000 will be visible or accessible. This is a compatibility access mode.

Note: Installing the Netware Client provides a new namespace called **NDS:.** This namespace is not related to the Windows NT Domain.

Installing the Internet Information Server is providing a new namespace called **IIS:**. This namespace is not related to the Windows NT Domain environment.

Note: Choosing a namespace implicitly defines the interfaces available for that namespace. Please refer to the Microsoft Active Directory Service Interfaces SDK for a complete list of supported interfaces in regards to the selected namespace. (Search for "Provider Support of ADSI Interfaces")

Using other credentials

Once you select a namespace to use, the security context used for the bind operation is determined. ADSI provides two main methods for binding to an object:

- **GetObject:** A client calls the **GetObject** function to bind to an ADSI object. It takes an LDAP path as input and returns a pointer to the requested interface. By default the binding uses a secure authentication option **with the security context of the current logged on user**. However, if the authentication fails, the secure bind is downgraded to an anonymous bind, for example, a simple bind without any user credentials.

- **OpenDSObject:** A client calls the **OpenDSObject** function to bind to an ADSI object using other credential than the current user security context. It takes an LDAP path as input and returns a pointer to the requested interface. The major advantages of using **OpenDSObject** are the following:

    - The ability to specify an alternate user name and password to authenticate to the directory.
    - The ability to use encryption to protect the data exchange over the network between application and the directory server
    - The possibility to specify the authentication method to use.

Use the logged on user's credentials whenever possible. However, if an application needs to supply alternate credentials, use **OpenDSObject** method.

The namespace used can have an influence in how the alternate credential is provided:

- Under the **WinNT:** namespace (Windows NT 4.0 and Windows 2000), use the form:

  "NTDomain\UserID"

- Under the **LDAP:** or **GC:** namespace (Windows 2000 only), use the form:

  "NTDomain\UserID"

- Under the **LDAP:** or **GC:** namespace (Windows 2000 only), use the form:

  "UserID@NTDomain.com"

- Under the **LDAP**: or **GC:** namespace (Windows 2000 only), use the form:

  "CN=UserID,CN=Users,DC=NTDomain,DC=com"

  Note: This form imposes a non-secure authentication.

- Under the **LDAP:** namespace (Exchange 5.5 only), use the form:

  "cn=UserID,dc=NTDomain"

Note: All samples listed in this document use the **GetObject** method. This means that the security context must have enough rights to run all the script properly. This approach was used to simplify the understanding and the script code reading. (See Sample 11 on page 39 at line 19 and 32 for a **GetObject** sample)

The Distinguished Name

The Active Directory may contain several NT domains, each of those NT domains contain several objects such as Computers, Users, Organizational Units, etc … Each of them has a position in the hierarchical organization of the Active Directory. So, to access any of the available objects, the programmer must be aware of their location in that structure. The location of an object is an LDAP path. As we have a path to access files on a disk, we have a similar concept for the Active Directory LDAP accesses.

The LDAP path is a type of fully qualified path name through the hierarchy of the Active Directory. It is often associated with the distinguished name. We will see that the distinguished name is sometimes a little bit different from the Active Directory real access path. It is mandatory to know this last property to access an object. Knowing only its name (or its relative distinguished name) as visible in most of the Windows 2000 interfaces is not enough to bind an application to the object.

Which syntax to use?

To access an object via the bind operation, an object path in a proper syntax must be provided. We will see that the syntax plays a role at two levels. It is important because the script will have to cover both levels.

- **ADsPath:** To obtain the correct path syntax related to a name space, use a property associated with each object by ADSI: **ADsPath**. This property returns a sort of distinguished Name. In fact, this property most of the time (but there are exceptions) has a value equal to the LDAP namespace pointer combined with the **distinguishedName**. This property is very useful when an application needs to browse inside the Active Directory. Each time an object container is discovered, it is possible to enumerate all of the objects it includes. Instead of building the LDAP path to these objects by programming, it can query the **ADsPath** property to have the exact path of objects listed in the container.

  Note: This property is not part of the schema. This property is built when getting information on the object (**GetInfo** method).

- **Attribute:** The syntax of an object attribute is determined by the definition stored in the Schema. Knowing the attribute syntax is important when reading or updating its values. A script must take care of this because ADSI returns value as Variant. It is important to know how to manipulate them. A script will never handle data *octetString* type as it will handle data *DirectoryString* type. Some tasks need to be executed to manipulate data correctly (type conversion and adapted function calls).

The Namespace syntax differences

To illustrate the namespace difference, a simple script will show the Administrator's group membership on a Windows 2000 Domain Controller.

Sample 11: Getting Administrator's group membership from a Windows 2000 DC in different namespaces.

```
' VB Script showing the Administrator's group membership via two  '
' different name spaces (WinNT: and LDAP) to show the name space  '
' syntax differences.                                             '
'                                                                 '
' Version 1.00 - Alain Lissoir                                    '
' Compaq Computer Corporation - Professional Services - Belgium - '
'                                                                 '
' Any comments or questions:       EMail:alain.lissoir@compaq.com '

Option Explicit

Dim objUser
Dim objMember
Dim strObjPath

' -----------------------------------------------------------------------------------------
First access via the WinNT: namespace
strObjPath = "WinNT://MyW2KDomain/Administrator"
Set objUser = GetObject(strObjPath)

WScript.Echo
WScript.Echo "ADSI Query via WinNT: namespace"
```

```
23:For Each objMember In objUser.Groups
24:    WScript.Echo "'" & objMember.Name & "' has a ADsPath of '" & objMember.AdsPath & "'"
25:Next
26:
27:Set objUser = Nothing
28:
29:' -------------------------------------------------------------------------------------------------
30:' Next access via the LDAP: namespace
31:strObjPath = "LDAP://CN=Administrator,CN=Users,DC=MyW2KDomain,DC=com"
32:Set objUser = GetObject(strObjPath)
33:
34:WScript.Echo
35:WScript.Echo "ADSI Query via LDAP: namespace"
36:For Each objMember In objUser.Groups
37:    WScript.Echo "'" & objMember.Name & "' has a ADsPath of '" & objMember.AdsPath & "'"
38:Next
39:
40:Set objUser = Nothing
```

This sample uses two different namespaces (**WinNT:** and **LDAP:**) to demonstrate the syntax differences. To keep it simple, the path to the Administrator object (line 18 and 31) is provided. In practice, the domain name of the Administrator should not be hard coded but it should be looked up via the **defaultDomainContext.** This will provide the exact domain distinguished name. This method is used in the Sample 10 on page 34.

When running this script, the following output is produced:

```
ADSI Query via WinNT: namespace
'Schema Admins' has a ADsPath of 'WinNT://w2k-home/Schema Admins'
'Enterprise Admins' has a ADsPath of 'WinNT://w2k-home/Enterprise Admins'
'Domain Admins' has a ADsPath of 'WinNT://w2k-home/Domain Admins'
'Domain Users' has a ADsPath of 'WinNT://w2k-home/Domain Users'
'Group Policy Admins' has a ADsPath of 'WinNT://w2k-home/Group Policy Admins'
'Administrators' has a ADsPath of 'WinNT://w2k-home/Administrators'

ADSI Query via LDAP: namespace
'CN=Group Policy Admins' has a ADsPath of 'LDAP://CN=Group Policy Admins,CN=Users,DC=w2k-
home,DC=com'
'CN=Domain Admins' has a ADsPath of 'LDAP://CN=Domain Admins,CN=Users,DC=w2k-home,DC=com'
'CN=Domain Users' has a ADsPath of 'LDAP://CN=Domain Users,CN=Users,DC=w2k-home,DC=com'
'CN=Enterprise Admins' has a ADsPath of 'LDAP://CN=Enterprise Admins,CN=Users,DC=w2k-home,DC=com'
'CN=Schema Admins' has a ADsPath of 'LDAP://CN=Schema Admins,CN=Users,DC=w2k-home,DC=com'
'CN=Administrators' has a ADsPath of 'LDAP://CN=Administrators,CN=Builtin,DC=w2k-home,DC=com'
```

The **WinNT:** namespace uses a very different syntax compared to the **LDAP:** namespace. It is important to use the correct path related to the namespace to access objects. This is important when writing an ADSI browser because this kind of application must be namespace independent. For such a case, ADSI provides the path and the syntax to use.

In addition to a classic naming variation caused by the presence of "CN=" ("DC=", or "o=") in some cases the pathname of the object itself changes. For instance when a name has a backslash (\) included, the name coding must be adapted. Under Windows 2000, a good example of a name with a slash is the subnet name associated with a site. When a subnet is created with the MMC plug-in "Active Directory Sites and Services", a name in the form: 200.200.200.0/24 must be provided. This means that the **distinguishedName** property is:

CN=200.200.200.0/24,CN=Subnets,CN=Sites,CN=Configuration,DC=w2k-home,DC=com

But the property **ADsPath** for that particular object is:

LDAP://CN=200.200.200.0\/24,CN=Subnets,CN=Sites,CN=Configuration,DC=w2k-home,DC=com

(Note the backslash before the /24)

This example shows the importance of using the **ADsPath** property of an object. If this property is not available, the script (or application) has to take care of this. In terms of string manipulation, this may complicate the programming.

Note: Active Directory Service Interfaces SDK includes specific methods to convert an **ADsPath** to a **distinguishedName.** Search for the **IADsPathname** interface in the Microsoft Active Directory Service Interfaces SDK for more information).

Another possible source to convert **ADsPath** to a **distinguishedName** is to use the Windows 2000 Resource Kit. It contains an IADSTOOLS.DLL. This .DLL exposes several functions to convert, extract and manipulate miscellaneous data coming from the Active Directory.

In a default Windows 2000 installation, there is another object that uses a slash in its name:

CN=DIRECTORY/MYW2KDC,CN=RpcServices,CN=System,DC=w2k-home,DC=com

The conclusion is that it is recommended to use the **ADsPath** for the bind operation. It should not be confused with the **distinguishedName** property. The **distinguishedName** property is not always usable to bind to objects. In the case of the last sample, this will create a binding error.

The object syntax differences

In the Active Directory, objects have different natures and different meaning. The way they are encoded in the Active Directory is adapted to the content. Each object in the Active Directory has a syntax. The syntax is located in the schema. By querying the syntax of an object's attribute, the script will get the necessary information to handle the returned value properly.

When manipulating values, it is important to know if the attribute is multi-valued or not. This means that the script must take in to account the fact that the attribute may have several values. In the case, the attribute is an array. Obviously, from the programming point of view, a single variable is not manipulated in the same way as an array.

The Active Directory provides all this information. The script must take this particularity in consideration.

Discovering the syntax of an object

As there is an **ADsPath** object property, there is also a Schema path property for an object. By using this property, it is possible to determine the attribute list associated with an object and the exact syntax of each associated attribute.

Sample 12: Accessing the syntax of an attribute

```
' VB Script showing the attributes syntax differences from a user object '
'                                                                         '
' Version 1.00 - Alain Lissoir                                            '
' Compaq Computer Corporation - Professional Services - Belgium -         '
'                                                                         '
' Any comments or questions:              EMail:alain.lissoir@compaq.com '

Option Explicit
.
' ---------------------------------------------------------------------------------------------------
' Get the default Windows 2000 Domain name
Wscript.Echo "Binding to RootDSE to get default Domain Name"
Wscript.Echo
```

```
21:Set objRoot = GetObject("LDAP://RootDSE")
22:strDefaultDomainName = objRoot.Get("DefaultNamingContext")
23:Set objRoot = Nothing
24:
25:' Create the Distinsguished Name (DN) by adding the default Windows 2000
26:' Domain name to the user name.
27:strUserDN = "CN=Administrator,CN=Users," & strDefaultDomainName
28:
29:' ---------------------------------------------------------------------------------------------------
30:' Bind to the user object
31:Wscript.Echo "Binding to '" & strUserDN & "'"
32:Wscript.Echo
33:Set objUser = GetObject("LDAP://" & strUserDN)
34:
35:' ---------------------------------------------------------------------------------------------------
36:' Bind to the class definition (User) path of the selected object.
37:Wscript.Echo "Binding to '" & objUser.Schema & "'"
38:Wscript.Echo
39:Set objUserClass = GetObject(objUser.Schema)
40:
41:' ---------------------------------------------------------------------------------------------------
42:Wscript.Echo "Show the 'sAMAccountName' syntax attribute of '" & strUserDN & "'"
43:
44:Set objProperty = GetObject(objUserClass.Parent + "/sAMAccountName")
45:WScript.Echo objProperty.Name & " (Syntax=" & objProperty.Syntax & ")" & _
46:                                " (Multi-Valued=" & objProperty.Multivalued & ")"
..:
51:' ---------------------------------------------------------------------------------------------------
52:Wscript.Echo "Show the 'objectSid' syntax attribute of '" & strUserDN & "'"
53:
54:Set objProperty = GetObject(objUserClass.Parent + "/objectSid")
55:WScript.Echo objProperty.Name & " (Syntax=" & objProperty.Syntax & ")" & _
56:                                " (Multi-Valued=" & objProperty.Multivalued & ")"
..:
..:
..:
```

In Sample 12 above, the user path is not explicitly provided. The RootDSE (lines 21 and 22) object knows the default Domain Name (**defaultNamingContext** attribute). This distinguished name is concatenated to the user name with respect to the syntax (line 27). By default Windows 2000 locates all the users in the CN=Users container. If a script needs to look after users created elsewhere in the Active Directory (i.e. in other “organizationalUnit”), the programmer has two choices:

- Knowing the correct **distinguishedName** property, a valid **ADsPath** can be built.
- A search using the ADO interface provided by ADSI. That method will suppress the need to know the **distinguishedName** property of an object because the query result will provide the answer.

As the bind to the user ‘Administrator’ object (line 33) is complete, it is possible to know what’s the schema path for this user. In this case, a user object is examined (known as a class ‘user’ object). This ‘user’ class is defined somewhere in the schema. The schema path is giving a pointer to the exact location of this class definition (line 39). The schema can be queried to get information about the syntax of two particular attributes: the **sAMAccountName** and the **objectSid** (lines 44 and 54). These two attributes are chosen in the example because there are using a totally different syntax. Running the script will have the following display output.

```
Binding to RootDSE to get default Domain Name
Binding to 'CN=Administrator,CN=Users,DC=w2k-home,DC=com'

Binding to 'LDAP://schema/user'

Show the 'sAMAccountName' syntax attribute of 'CN=Administrator,CN=Users,DC=w2k-home,DC=com'
sAMAccountName (Syntax=DirectoryString) (Multi-Valued=False)
```

```
Show the 'objectSid' syntax attribute of 'CN=Administrator,CN=Users,DC=w2k-home,DC=com'
objectSid (Syntax=OctetString) (Multi-Valued=False)
```

As can be seen, the **sAMAccountName** attribute is a *DirectoryString* and the **objectSid** attribute is an *OctetString*. Both attributes are single valued. When handling those attribute values in a script, the developer must be aware of their respective types.

Discovering the Directory Tree, Objects, attributes and syntaxes

Now we are able to find some attributes from the Active Directory. Looking from the **RootDSE** object, we know how to access an object and its attributes. What about all the other objects? What about their attributes and their syntaxes? What about unknown objects present in the Active Directory? Is there documentation about this? The answer to all of those questions is same: **“The information is located in the Active Directory Schema”.**

The solution is to have a script able to extract the interesting information from the Active Directory Schema. The script should be able to determine if an object has attributes and for those attributes what’s the associated syntax. This is the purpose of the script Sample 13 below. This will help to determine what needs to be done to access the desired data.

To do so, the **MandatoryProperties** and **OptionalProperties** methods are used from any object class definition. Those two ADSI methods return the list of all available attributes for an object class. This script is exactly the same as Sample 12. The only difference is that two new loops are inserted. The first loop is to browse the Active Directory through a recursive routine looking if there are some available objects inside the examined object (container) (lines 78 to 97). The second loop is to list all the mandatory and optional attributes defined in the schema for the object’s class (lines 117 to 133).

The script does not display the results at the Command Prompt. You must have Excel installed on the machine where you intend to run this script. The results of the browsing operation are loaded into an Excel sheet. The use of Excel makes the output easier to read because a lot of data is produced.

Sample 13: Loading the entire Active Directory Tree objects with their attributes and syntaxes loaded in an Excel sheet.

```
 1:' VB Script loading all objects from an AD context location into an Excel sheet '
 2:' (with attributes and syntaxes)                                                  '
 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:' Set this constant to zero, if you don't want indentation in the Excel Sheet
12:Const cIndent = 1
..:
21:' ----------------------------------------------------------------------------------------------------
22:' Start the Excel Worksheet reading
23:Public objXL
24:
25:' Bind to an Excel worksheet object
26:Set objXL = WScript.CreateObject("EXCEL.application")
..:
37:' ----------------------------------------------------------------------------------------------------
38:intY = 0
39:intX = 0
..:
61:WScript.Echo strObject
62:objXL.activecell.offset(intY, intX).Value = strObject
```

```

Call LookInsideObject ("LDAP://" & strObject , intX + cIndent)
..
WScript.Quit (0)

' -------------------------------------------------------------------------------------
Private Sub LookInsideObject (strObject, intX)

Dim objObject
Dim objObjectClass
Dim objMember

        Set objObject = GetObject(strObject)
        Set objObjectClass = GetObject(objObject.Schema)

        WScript.Echo Space (intX) & objObject.Name
        Call GetMemberInfo (objObject, objObjectClass, intX)

        For Each objMember in objObject
            Call LookInsideObject (objMember.ADsPath, intX + cIndent)
        Next

        Set objObjectClass = Nothing
        Set objObject = Nothing

End Sub

' -------------------------------------------------------------------------------------
Private Sub GetMemberInfo (objObject, objObjectClass, intX)

        intY = intY + 1

        objXL.activecell.offset(intY, intX).Value = objObject.Name
        objXL.activecell.offset(intY, intX + 1).Value = objObject.Class
        objXL.activecell.offset(intY, intX + 2).Value = objObject.ADsPath

        ' Show object's mandatory attributes with syntax
        Call LoadPropertiesInXL (objObjectClass.MandatoryProperties, objObjectClass, intX)

        ' Show object's optional attributes with syntax
        Call LoadPropertiesInXL (objObjectClass.OptionalProperties, objObjectClass, intX)

End Sub

' -------------------------------------------------------------------------------------
Private Sub LoadPropertiesInXL (PropertyList, objObjectClass, intX)

Dim strProperty
Dim objProperty

        For Each strProperty in PropertyList
            Set objProperty = GetObject(objObjectClass.Parent + "/" + strProperty)

            intY = intY + 1
            objXL.activecell.offset(intY, intX).Value = objProperty.Name
            objXL.activecell.offset(intY, intX + 1).Value = objProperty.Syntax
            objXL.activecell.offset(intY, intX + 2).Value = objProperty.Multivalued

            Set objProperty = Nothing
        Next

End Sub
...
...
```

The key to the script resides in the recursive effect. Each time a bind to an object is done (after displaying its attributes with their syntax at line 88) a “For Each” loop (line 90) is performed to determine if this object contains other objects. This is very important because the recursive loop gives an answer to a fundamental question when browsing the Active Directory: **Is the object a container?** This information is also available from an object attribute but it is not necessary to query that characteristic, as the “For Each” loop will take care of this.

The code is somewhat less readable than before. This is due to the Excel COM object references needed to load the huge volume of information retrieved. If this script is run in a default Windows 2000 installation, it will produce a very manageable Excel sheet size (± 500K). Of course, **it is not recommended to use this script in a production environment** with thousands of users and servers. The purpose of the Sample 13 is to help find the information needed during the script development phase.

By default, Sample 13 looks for the distinguished name of the **defaultNamingContext**. If no parameter is specified on the command line, the script will prompt the user for a distinguished name start point. This script will act transparently and will get a list of the available objects with their attributes and associated syntax. This “low cost browser” can be very useful. The data contained in the Excel sheet was used to write all the samples in this document

Below is a partial screen output from running the script. Each time a container is found by the “For Each” loop the script what the object it includes.

```
DC=dev,DC=w2k-home,DC=com
 DC=dev
  CN=Builtin
   CN=Account Operators
   CN=Administrators
   CN=Backup Operators
   CN=Guests
   CN=Print Operators
   CN=Replicator
   CN=Server Operators
   CN=Users
  CN=Computers
  OU=Domain Controllers

… continue …
```

Getting and setting object’s attribute values in respect of the syntax

Once the syntax (*DirectoryString*, *OctetString*, …) and the value attribute form (*Multi-valued*, *single-valued*) are known, some retrieval-adapted methods are used depending on theses characteristics. Unfortunately, we are not living in a perfect world. There is still a last tricky characteristic to deal with. Some objects, as the User Object, have three specific attribute types:

(Extracted from the ADSI SDK)

- **Domain-replicated, stored attributes**
  Some attributes are stored in the directory (such as **cn**, **nTSecurityDescriptor**, **objectGUID**, …) and replicated to all domain controllers within a domain. A subset of these attributes is also replicated to the global catalog. If a script enumerates the attributes of a user object from the global catalog, only the attributes that are replicated to the global catalog are returned. Some attributes are also indexed. Including an indexed attribute in a query improves its performance.

- **Non-replicated, locally stored attributes**
  Non-replicated attributes are stored on each domain controller but are not replicated elsewhere (such as **badPwdCount**, **lastLogon**, **lastLogoff**, and so on). The non-replicated

attributes are a attributes that pertain to a particular domain controller. For example, **lastLogon** is the last date/time that the user's network logon was validated by the particular domain controller that is returning the attribute. These attributes can be retrieved in the same way as the domain-wide attributes described previously. However, for these attributes, each domain controller stores only values that pertain to that particular domain controller. For example, to get the last time a user logged on to the domain, the **lastLogon** attribute for the user at every domain controller in the domain has to be read, the times compared, and then the latest time can be found.

- **Non-stored, constructed attributes**
  A user object also has constructed attributes that are not stored in the directory but are calculated by the domain controller (such as **canonicalName**, **distinguishedName**, **allowedAttributes**, **ADsPath**, …). Most are automatically retrieved and cached when getting attribute value on the object. However, some constructed attribute are not set and therefore require a specific method to explicitly retrieve them. For example, the **canonicalName** is not retrieved by a traditional Get method. The attribute is constructed when invoking the GetInfoEx method.

The attribute characteristic is also located in the Active Directory Schema. An **attributeSchema** definition (which is nothing more than the definition of an object located in the Schema context) has an attribute called **systemFlag**. That is where this supplemental information is stored. This means that the script must bind to the **attributeSchema** related to the attribute for which this information has to be retrieved.

The layout of the **systemFlag** is explained in the ADS_SCHEMA_SYSTEMFLAG enumeration defined in the Microsoft Active Directory Services Interfaces SDK. Here are the existing values and their meaning:

- ADS_SYSTEMFLAG_DISALLOW_DELETE
  The attribute cannot be deleted.
- ADS_SYSTEMFLAG_CONFIG_ALLOW_RENAME
  The configuration attribute can be renamed.
- ADS_SYSTEMFLAG_CONFIG_ALLOW_MOVE
  The configuration attribute can be moved.
- ADS_SYSTEMFLAG_CONFIG_ALLOW_LIMITED_MOVE
  The configuration attribute can be moved with restrictions.
- ADS_SYSTEMFLAG_DOMAIN_DISALLOW_RENAME
  The domain attribute cannot be renamed.
- ADS_SYSTEMFLAG_DOMAIN_DISALLOW_MOVE
  The domain attribute cannot be moved.
- ADS_SYSTEMFLAG_CR_NTDS_NC
  Naming context is in NTDS.
- ADS_SYSTEMFLAG_CR_NTDS_DOMAIN
  Naming context is a domain.
- ADS_SYSTEMFLAG_ATTR_NOT_REPLICATED
  The attribute is not to be replicated.
- ADS_SYSTEMFLAG_ATTR_IS_CONTRUCTED
  The attribute is a constructed attribute.

To acquire the **systemFlag**, Sample 13 on page 43 is not sufficient. The next part of this study will complete Sample 13 to provide the missing information. For this, a search function is needed. The advanced part of the study shows how to build a search function and how to use it to complete this set of information coming from the Active Directory Schema. (See Part 2: "The powerful combination of WSH and ADSI under Windows 2000") The purpose is to get the right attribute in the right way.

Which methods to use to manipulate ADSI data?

To summarize, getting an attribute value from an object in the Active Directory requires knowing several things:

- The Active Directory context of the desired object. (**DefaultNaming Context**, **SchemaNaming Context**, **ConfigurationNaming Context**)
- The Active Directory location of the desired object in the selected context (**ADsPath**).
- The object attribute name desired. (**distinguishedName**, **canonicalName, lastName**)
- The syntax of the desired attribute. (OctetString, DirectoryString)
- The value type of the desired attribute. (Multi-valued)
- The attribute type. (Domain-replicated, stored attributes, Non-replicated, locally stored attributes, Non-stored, constructed attributes)

Once characteristics of the objects and their related attributes are known, it is possible to access the information in respect of that information. With the ***IADs*** interface, ADSI provides several methods to get or set values from/to Active Directory objects:

- **Get Method**

Individual attributes can be retrieved from the directory using the Get method.

```
Value = Object.Get ("attributeName")
```

With automation languages, the attribute name can be directly used with the dot notation if the ADSI COM object implements a property for it:

```
Value = Object.COMpropertyName
```

- **GetEx Method**

Some attributes are returned as multiple values. They can contain one or more values. For instance, a list of descriptions on a domain is a multi-valued attribute. A multi-valued attribute can be retrieved as an array using the GetEx method.

```
objList = Object.GetEx("MultiValuedPropetyName")

For Each Element In objList
    Wscript.Echo Element
Next
```

GetEx gets attributes that support single or multiple values in variant structures from the property cache.

- **GetInfo Method**

The GetInfo method is used to refresh ADSI object's cached attributes from the underlying directory service. ADSI invokes an implicit GetInfo if a Get is performed on a specific attribute in the property cache and no value is found. Once GetInfo has been called, an implicit call will not be repeated. If a value already exists in the property cache, however, calling **Get** without first calling GetInfo will retrieve the cached value rather than the most current value from the underlying directory. To obtain the most recent values for an object, always call GetInfo. Any changes made in the property cache will be replaced with the current values from the server.

```
Object.GetInfo
```

- **GetInfoEx Method**

The GetInfoEx method is called explicitly to refresh some ADSI object's cached properties from the underlying namespace. To refresh all properties, use GetInfo. GetInfoEx gets specific current values for the attributes of an Active Directory object from the underlying directory store, refreshing the cached values.

```
Object.GetInfoEx Array("description", "distinguishedName", “canonicalName”), 0
```

After this call, the cache reflects the attribute values in the underlying namespace directory store. However, the cache is only updated with the values specifically requested in the GetInfoEx call.

Some servers will not return all attributes of an object in response to a GetInfo call. In this case, an explicit GetInfoEx call naming these "non-default" attributes (such as the “Non-stored, constructed attributes”) has to be done in order to get them in the cache. As shown in the example, GetInfoEx is the only way to get the **canonicalName** attribute in the cache because it is a constructed attribute.

- **Put Method**

The Put method saves the value of a named Active Directory object attribute into the property cache. This value is not updated in the underlying directory service until SetInfo is called.

```
Object.Put “description”, strDescription
```

- **PutEx Method**

The PutEx method uses the name of an attribute to save a single or multi-valued attribute into the property cache. This overwrites any value in the property cache. The values in the cache are not written to the underlying directory service until a SetInfo occurs. The first argument of PutEx indicates whether you want to clear, update, add or append to any existing attribute value.

```
Const ADS_PROPERTY_CLEAR  = 1
Const ADS_PROPERTY_UPDATE = 2
Const ADS_PROPERTY_APPEND = 3
Const ADS_PROPERTY_DELETE = 4

Object.PutEx ADS_PROPERTY_APPEND, "siteList", Array(strSiteOne, strSiteTwo)
```

In the example, two new sites are appended to the existing **siteList** attribute.

- **SetInfo Method**

The SetInfo method saves the current object attribute values from the property cache to the underlying directory store. This is analogous to flushing a buffer out to disk.

SetInfo will update objects that already exist in the directory or create a new directory entry for newly created objects.

At the time of the SetInfo call, if any property cache values were written with a PutEx control code such as ADS_PROPERTY_UPDATE or ADS_PROPERTY_CLEAR, then the appropriate requests are passed on to the underlying directory service.

```
Object.SetInfo
```

Important ! SetInfo is always an explicit call. It is never called implicitly. It is the programmer's responsibility to invoke that method at the right time to commit all the changes to the underlying directory.

COMPLETING THE WSH FUNCTIONS SET WITH ADSI INFORMATION

Once a Logon Script is started by the Windows 2000 Group Policies configurations, it can be useful to enhance the set of information available from the scripting environment, WSH. ADSI is able to provide in the script run-time environment many other Active Directory data.

The information available from ADSI complementary data set can be about the user currently executing the Logon script (i.e. Full Name, Email address), but also about the current site location of this user. The following section explains how to access and use this information from a logon script

Getting the user's distinguished name

Getting the user's distinguished name from a Logon script is a particular case because it needs to locate the user object in the Active Directory to know its distinguished name. This is a chicken and egg problem. "How do I know the user's distinguished name if the script is not yet bound to the user object, whereas the script needs at least to know the distinguished name to make the bind operation?"

Of course, the script developer can assume the default user object location in the CN=Users container. But this is not realistic because any user object can be located in any other container (such as organizational units). Assuming the default will never be usable in practice.

Another difficulty is to know the username from the script environment. The only thing you know from such context is the username provided by the environment variable or the username provided by the *Wnetwork* object (See Sample 1 on page 15). Under Windows 2000, a user can logon by using the down level client form (DomainName\UserName) or the UPN (User Principal Name) form (firstname.lastname@DomainName.Com)

A solution is to launch a query from the script in the Active Directory (or in the Global Catalog) to locate the user object using this down level or UPN name. This will seriously complicate the Logon script development.

To address the problem, Microsoft provides an ADSI interface called *IADsSystemInfo*. This object exposes attributes providing useful information about the current user logged in the system.

Note: ADSI 2.5 does not expose the *IADsSystemInfo* interface. This is an ADSI object only available under Windows 2000.

Sample 14 Getting the user distinguished name

```
...
...
...
' Bind to the IADsSystemInfo to get current system and user information. --------------------------
Dim objSysinfo

Set objSysinfo = CreateObject("ADSystemInfo")
...
strUserNameDN = objSysinfo.UserName
strUserNameADsPath = "GC://" & strUserNameDN
strLogonSiteName = objSysinfo.SiteName
...
```

As any regular object, the script must first instantiate the ADSI object (line 222). Next, the miscellaneous properties are available to extract the user's distinguished name and the site name. Note the construction of the **ADsPath**. Any other reference to the user object will use the Global Catalog instead of the local Active Directory (LDAP:). Because the script needs to get information about users from any domain in the organization, the Global Catalog is the only database containing information about all users in the organization. (For this, desired attributes need to be replicated in the GC, See the explanation of **systemFlags** on page 45).

Sample 15 Information available from the *IADsSsystemInfo* interface.

```
  .:
  .:
  .:
  9:Dim objSysInfo
 10:
 11:Set objSysInfo = CreateObject("ADSystemInfo")
 12:
 13:Wscript.Echo "UserNameDN=" & objSysInfo.UserName
 14:Wscript.Echo "ComputerName=" & objSysInfo.ComputerName
 15:Wscript.Echo "SiteName=" & objSysInfo.SiteName
 16:Wscript.Echo "DomainShortName=" & objSysInfo.DomainShortName
 17:Wscript.Echo "DomainDNSName=" & objSysInfo.DomainDNSName
 18:Wscript.Echo "ForestDNSName=" & objSysInfo.ForestDNSName
 19:Wscript.Echo "PDCRoleOwner=" & objSysInfo.PDCRoleOwner
 20:Wscript.Echo "SchemaRoleOwner=" & objSysInfo.SchemaRoleOwner
 21:Wscript.Echo "IsNativeMode=" & objSysInfo.IsNativeMode
 22:Wscript.Echo "GetAnyDCName=" & objSysInfo.GetAnyDCName
 23:Wscript.Echo "GetDCSiteName=" & objSysInfo.GetDCSiteName ("W2K-DPEN6400DEV")
 ..:
 ..:
 ..:
```

Getting the user Fullname

Once the user's distinguished name is known, it is possible to execute a bind operation on the user object. Any information replicated to the Global Catalog and available from the user object can be read. When the logon script makes a popup window to show the number of unread mail in his mailbox, the script shows the user's full name (See Sample 7 on page 23). The user's full name is read in Sample 16.

Sample 16 Getting the user Fullname

```
...:
...:
...:
231:' -----------------------------------------------------------------------------------------------
232:' Get the user FullName stored in AD via ADSI
233:strUserFullName = GetUserFullName (objLogFileName, strUserNameADsPath)
...:
481:' -----------------------------------------------------------------------------------------------
482:Private Function GetUserFullName (objFileName, strUserNameADsPath)
...:
491:        Set objUser = GetObject(strUserNameADsPath)
...:
497:        strUserFullName = objUser.Get("Name")
...:
504:        GetUserFullName = strUserFullName
505:        Set objUser = Nothing
506:
507:End Function
...:
...:
...:
```

Group Membership checking

For any other regular Logon script, it is important to know the current user membership to determine assignments or operations to be made (i.e. Network resource connections). From a native WSH object it is impossible to determine this information.

With ADSI, the *IADsUser* interface offers a method to know the group membership of a given user (See Sample 11 on page 39). The principle is to encapsulate this method invocation (line 853) in a function returning a Boolean value. If the user is member of the given group, True is returned, otherwise False is returned. This is shown in Sample 17.

Sample 17 Group Membership checking

```
...:
...:
...:
267:If GroupMember (objLogFileName, strUserNameADsPath, "Domain Admins") Then
268:
269:   ' ---------------------------------------------------------------------------------------------
270:   ' Current user is a 'Domain Admins' member, makes special mappings for him
...:
276:End If
...:
835:' ------------------------------------------------------------------------------------------------
836:Private Function GroupMember (objFileName, strUserNameADsPath, strGroupName)
837:
838:Dim objUser
839:Dim objGroup
...:
847:        Set objUser = GetObject(strUserNameADsPath)
...:
853:        For Each objGroup In objUser.Groups
854:            If strGroupName = objGroup.Get("Name") Then
...:
856:               GroupMember = True
857:               Exit Function
858:            End If
859:        Next
...:
862:        GroupMember = False
863:
864:End Function
...:
...:
...:
```

Getting the Default Domain or the Root Domain distinguished name

For Sample 10 on page 34, important information to have inside a Logon script is the Default Domain or the Root Domain distinguished name. The method is not different from the previous sample except that the operation is encapsulated in a function.

Sample 18 Getting the Default Domain or the Root Domain distinguished name

```
...
...
...
' -----------------------------------------------------------------------------------------
' Determine distinquished names and ADsPath needed for further object bindings with ADSI
...
strRootDomainNameDN = GetRootDomainNameDN (objLogFileName)
...
strDefaultDomainNameDN = GetDefaultDomainNameDN (objLogFileName)
...
' -----------------------------------------------------------------------------------------
Private Function GetRootDomainNameDN (objFileName)

Dim objRoot
Dim strRootDomainContext
Dim objRootDomainContext
...
        Set objRoot = GetObject("LDAP://RootDSE")
        strRootDomainContext = objRoot.Get("RootDomainNamingContext")
        Set objRootDomainContext = GetObject("LDAP://" & strRootDomainContext)
...
        GetRootDomainNameDN = objRootDomainContext.distinguishedName

        Set objRootDomainContext = Nothing
        Set objRoot = Nothing

End Function
...
' -----------------------------------------------------------------------------------------
Private Function GetDefaultDomainNameDN (objFileName)

Dim objRoot
Dim strDefaultDomainContext
Dim objDefaultDomainContext
...
        Set objRoot = GetObject("LDAP://RootDSE")
        strDefaultDomainContext = objRoot.Get("DefaultNamingContext")
        Set objDefaultDomainContext = GetObject("LDAP://" & strDefaultDomainContext)
...
        intRC = WriteToFile (objFileName, objDefaultDomainContext.distinguishedName)

        GetDefaultDomainNameDN = objDefaultDomainContext.distinguishedName

        Set objDefaultDomainContext = Nothing
        Set objRoot = Nothing

End Function
...
...
...
```

Logon Script Sample

The Logon Script sample is provided in the Sample Script kit accompanying this White Paper. It contains the full listing (and usable) scripts. To give an overview of all functions available for reuse, Sample 19 gives an overview of all function definitions with their parameters.

Sample 19 Logon Script Sample functions overview

```
Private Function ReadRegistry (objFileName, strKeyName, KeyValueName, strRegType)
Private Function WriteRegistry (objFileName, strKeyName, KeyValueName, strRegType, varRegKeyValue)
Private Function DeleteRegistry (objFileName, strKeyName, KeyValueName)
Private Function GetUserFullName (objFileName, strUserNameADsPath)
Private Function EnumerateDriveConnections (objFileName)
Private Function ConnectNetworkDrive (objFileName, strDriveLetter, strShareName)
Private Function DisconnectNetworkDrive (objFileName, strDriveLetter, boolConfirm)
Private Function EnumeratePrinterConnections (objFileName)
Private Function ConnectWindowsNetworkPrinter (objFileName, strShareName, boolDefault)
Private Function ConnectNetworkPrinter (objFileName, strLPT, strShareName)
Private Function DisconnectNetworkPrinter (objFileName, strLPT, boolConfirm)
Private Function ReadEnvironmentVariable (objFileName, strEnvironmentType, strVarName)
Private Function CreateEnvironmentVariable (objFileName, strEnvironmentType, strVarName, varValue)
Private Function RemoveEnvironmentVariable (objFileName, strEnvironmentType, strVarName)
Private Function GetAllEnvironmentVariables (objFileName, strEnvironmentType)
Private Function ReadCommandLineArgument (objFileName, strParameterList)
Private Function GroupMember (objFileName, strUserNameADsPath, strGroupName)
Private Function GetRootDomainNameADsPath (objFileName)
Private Function GetRootDomainNameDN (objFileName)
Private Function GetDefaultDomainNameADsPath (objFileName)
Private Function GetDefaultDomainNameDN (objFileName)
Private Function CheckMAPIMail (objFileName, strMAPIProfileName, strUserName, boolPrompt)
Private Function LogPublicVariables (objFileName)
Private Function ErrorHandler (objFileName, strFunctionName, Err, boolPopupErrors)
Private Function CreateTextFile (strFileName)
Private Function CloseTextFile (objFileName)
Private Function WriteToFile (objFileName, Text)
```

CONCLUSION

Windows Script Host as provided under Windows 2000 is powerful. The power of this scripting environment comes mainly from the COM object reusability.

In such an environment, the knowledge of a scripting language (such as Visual Basic Script or Java Script) is required but the challenge is not to know the programming language. Today, the challenge resides more in the existing COM objects knowledge. Knowing the COM available methods and properties is the key to successfully control the power of Windows 2000.

In this document, ADSI interfaces are mainly used. Today, Microsoft provides to the script developer most of the Windows Operating System parts through COM objects (such as WMI, MAPI or Internet Explorer interfaces for instance).

For a long time, Windows has suffered from a lack of scripting functionalities. Today, with Windows Script Host and all COM objects available under Windows 2000, administrators and developers receive the right answer. Windows Script Host and the COM objects functionalities fill a gap between the simple batch file development and advanced MFC programming.

APPENDIX A: ADSI INTERFACES LIST

Figure 7 ADSI Interfaces for each name space used.

Interface Name	LDAP	WinNT	NDS	NWCOMPAT
IADs	Yes	Yes	Yes	Yes
IADsAccessControlEntry	Yes	No	Yes	No
IADsAccessControlList	Yes	No	Yes	No
IADsAcl	No	No	Yes	No
IADsBackLink	No	No	Yes	No
IADsCaseIgnoreList	No	No	Yes	No
IADsClass	Yes	Yes	Yes	Yes
IADsCollection	No	Yes	No	Yes
IADsComputer	No	Yes	No	Yes
IADsComputerOperations	No	Yes	No	Yes
IADsContainer	Yes	Yes	Yes	Yes
IADsDeleteOps	Yes	No	No	No
IADsDomain	No	Yes	No	No
IADsEmail	No	No	Yes	No
IADsExtension	Yes	Yes	No	Yes
IADsFaxNumber	No	No	Yes	No
IADsFileService	No	Yes	No	Yes
IADsFileServiceOperations	No	Yes	No	Yes
IADsFileShare	No	Yes	No	Yes
IADsGroup	Yes	Yes	Yes	Yes
IADsHold	No	No	Yes	No
IADsLargeInteger	Yes	No	No	No
IADsLocality	Yes	No	Yes	No
IADsMembers	Yes	Yes	Yes	Yes
IADsNamespaces	Yes	Yes	Yes	Yes
IADsNetAddress	No	No	Yes	No
IADsO	Yes	No	Yes	No
IADsOU	Yes	No	Yes	No
IADsObjectOptions	Yes	No	No	No
IADsOctetList	No	No	Yes	No
IADsOpenDSObject	Yes	Yes	Yes	No
IADsPath	No	No	Yes	No
IADsPathname	Yes	Yes	Yes	Yes
IADsPostalAddress	No	No	Yes	No
IADsPrintJob	No	Yes	No	Yes
IADsPrintJobOperations	No	Yes	No	Yes
IADsPrintQueue	Yes	Yes	Yes	Yes
IADsPrintQueueOperations	Yes	Yes	Yes	Yes
IADsProperty	Yes	Yes	Yes	Yes
IADsPropertyEntry	Yes	Yes	Yes	Yes
IADsPropertyList	Yes	Yes	Yes	Yes
IADsPropertyValue	Yes	Yes	Yes	Yes

Interface Name	LDAP	WinNT	NDS	NWCOMPAT
IADsPropertyValue2	Yes	Yes	Yes	Yes
IADsReplicaPointer	No	No	Yes	No
IADsResource	No	Yes	No	No
IADsSecurityDescriptor	Yes	No	Yes	No
IADsService	No	Yes	No	No
IADsServiceOperations	No	Yes	No	No
IADsSession	No	Yes	No	No
IADsSyntax	Yes	Yes	Yes	Yes
IADsTimestamp	No	No	Yes	No
IADsTypedName	No	No	Yes	No
IADsUser	Yes	Yes	Yes	Yes
IDirectoryObject	Yes	No	Yes	No
IDirectorySearch	Yes	No	Yes	No

APPENDIX B: REFERENCES AND POINTERS

Microsoft WSH

- Windows Script Host 2.0 (On-line HTML help)
  http://msdn.microsoft.com/scripting/windowshost/wshdoc.exe
- Windows Script Host Download
  http://www.microsoft.com/scripting/downloads/ws/x86/ste51en.exe (WSH 2.0)
  http://www.microsoft.com/scripting/downloads/ws/x86/scr55en.exe (WSH 5.5 for Windows NT 4.0)
  http://www.microsoft.com/scripting/downloads/ws/x86/scripten.exe (WSH 5.5 for Windows 2000)
  The following files will install Windows Script Components containing:
  - Visual Basic Script Edition (VBScript) Version 5.1 or 5.5.
  - JScript® Version 5.1 or 5.5.
  - Windows Script Components
  - Windows Script Host 2.0
  - Windows Script Runtime Version 5.1 or 5.5.
- Windows Script Components (Documentation)
  http://msdn.microsoft.com/scripting/scriptlets/serverdocs.htm
- Windows Script Components (On-line HTML help)
  http://msdn.microsoft.com/scripting/scriptlets/wscdoc.exe
- Windows Script Component Wizard
  http://msdn.microsoft.com/scripting/scriptlets/wz10en.exe
- Microsoft Script Encoder
  http://msdn.microsoft.com/scripting/vbscript/download/x86/sce10en.exe

Windows Script Host Editor

- PrimalSCRIPT 2.0 - The Windows Script Host Editor
  http://www.sapien.com/PrimalSCRIPT.htm

Microsoft Scripting

- Windows Script Debugger
  http://msdn.microsoft.com/scripting/debugger/default.htm
- VB Script Documentation and Java Script Documentation
  http://msdn.microsoft.com/scripting/vbscript/techinfo/vbsdocs.htm
  http://msdn.microsoft.com/scripting/jscript/techinfo/jsdocs.htm
- File System Object Tutorial
  http://msdn.microsoft.com/scripting/vbscript/doc/jsFSOTutor.htm
  http://msdn.microsoft.com/scripting/jscript/doc/jsFSOTutor.htm

Internet sites about Scripting

- Win 32 Scripting
  http://cwashington.netreach.net/
- WSH Glazier Systems
  http://wsh.glazier.co.nz/default.asp
- The WinScripter
  http://www.winscripter.com/index.html

Microsoft ADSI

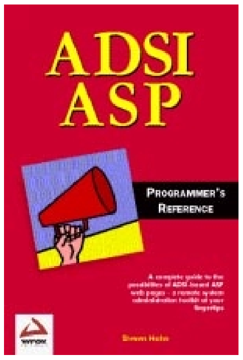
- Microsoft ADSI
  http://www.microsoft.com/windows/server/Technical/directory/adsilinks.asp
- Microsoft ADSI 2.5 Download
  http://www.microsoft.com/ntserver/nts/downloads/other/ADSI25/default.asp
- Microsoft ADSI 2.5 SDK Download
  http://www.microsoft.com/ntserver/nts/downloads/other/ADSI25/sdk.asp
- Active Directory Service Interfaces (ADSI) Implemented in Java
  http://www.microsoft.com/ntserver/nts/downloads/previews/NTSADSIJava/default.asp
- Microsoft News Group
  nntp://microsoft.public.active.directory.interfaces

Internet Sites about ADSI

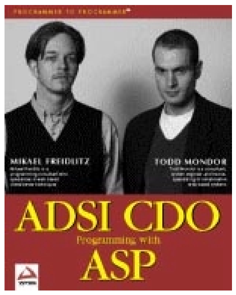
- 15 Seconds (ADSI)
  http://www.15seconds.com/focus/ADSI.htm

Books about ADSI

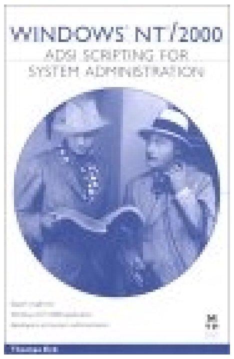
- ADSI ASP Programmer's Reference
  http://www.wrox.com/Consumer/Store/Details.asp?ISBN=186100169X



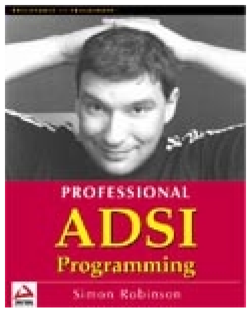
- Professional ADSI CDO Programming with ASP
  http://www.wrox.com/Consumer/Store/Details.asp?ISBN=1861001908



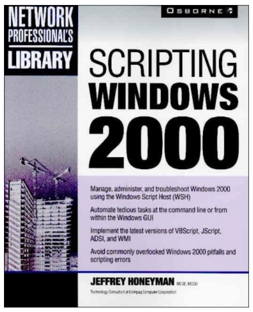
- Windows NT/2000 ADSI Scripting for System Administration
  http://www.amazon.com/exec/obidos/ASIN/1578702194/ref=sim_books/102-2951111-8168957



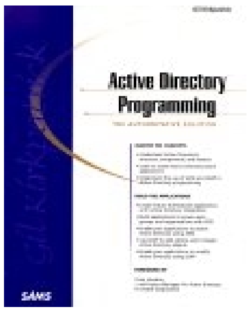
- Professional ADSI Programming
  http://www.wrox.com/Consumer/Store/Details.asp?ISBN=1861002262



- Scripting Windows 2000 (Compaq Computer)
 http://www.amazon.com/exec/obidos/ASIN/007212444X/ref=sim_books/104-4077239-3243961

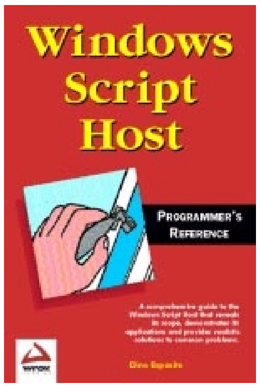


- Professional ADSI Programming
 http://www.amazon.com/exec/obidos/ASIN/0672315874/qid%3D967880578/102-2951111-8168957



Books about WSH

- Windows Script Host Programmer's Reference
 http://www.wrox.com/Consumer/Store/Details.asp?ISBN=1861002653



- Windows Script Host
  http://www.amazon.com/exec/obidos/ASIN/1578701392/qid%3D953064013/103-4505117-1055804