4

WMI Security Scripting

4.1 Objective

Previously, we discovered how to manage various components of Windows. In some cases, we saw that the security configuration is part of the component management. In this chapter, we will discover the WMI capabilities to manage the security settings of various Windows components, such as files, folders, and shares on the file system; Active Directory objects; and CIM repository namespaces. Although quite specific, the manipulation of the security settings, defined by security descriptors, is one of the most complex tasks to script. This chapter will explain the security descriptor components, their roles, and how to decipher them. One of the goals is to help you navigate between the various challenges that you face when automating and maintaining the security configuration under Windows. Beyond that, we will also see the security implication when developing ASP WMI-enabled scripts for Internet Information Server and how the Microsoft security push initiative in early 2002 affects WMI scripting under Windows Server 2003.

4.2 The WMI security configuration

Any manageable object we discussed in the previous chapters can be accessed under some security conditions. The WMI access is defined by three methods:

- During the WMI connection, the entity accessing a system must provide an authentication method and some privileges to perform specific tasks (i.e., system reboot) or access some specific manageable objects (i.e., security event log).
- The entity accessing the manageable object is granted access to a CIM repository namespace and allowed to perform some specific opera-

tions. The entity in question could also be part of an authorized group.

- The manageable object is restricted in access in a namespace by the means of a security descriptor. A security descriptor is nothing other than a representation, in the form of a list, of the access rights granted or denied to entities for the purpose of accessing a secured object in the system.

4.2.1 The WMI connection security settings

The establishment of a WMI connection always includes an authentication method, an impersonation level, and some optional privileges definition. Throughout the first book, *Understanding WMI Scripting*, we talk about this aspect when we explain how to perform a WMI connection. You can refer to Chapter 4 of the first book, sections 4.3.3, "The security settings of the moniker," and 4.1, "Establishing the WMI connection," for more information about the WMI connection scripting technique and the various connection settings.

However, it is important to determine the type of privileges required for a specific WMI operation. Table 4.1 lists the WMI classes that require some specific privileges at the class level, the property level, or the method level.

4.2.2 The group membership

By default, every CIM repository namespace is secured from the **Root** namespace, while subnamespaces inherit the security settings from the **Root** (see Figure 4.1).

By default, only two built-in groups are configured to access a CIM repository namespace: *Administrators* and *Everyone*. While the *Administrators* group has full access to any CIM repository namespace by default, the *Everyone* group is restricted to a limited number of accesses, such as reading some configuration data from the local system.

Table 4.2 summarizes the access type granted to the default groups for the **Root** namespace. Each of these rights is stored in the CIM repository in the form of a security descriptor.

Table 4.1 *The WMI Privileges Required for Some Classes, Properties, or Methods*

Namespace	WMI Class	CIM Element	Name	Privileges	Description
ROOT/CIMV2	Win32_PageFileUsage	Class		SeCreatePagefilePrivilege	The Win32_PageFileUsage class represents the file used for handling virtual memory file swapping on a Win32 system. Information contained within objects instantiated from this class specify the runtime state of the page file. **Note:** The SE_CREATE_PAGEFILE privilege is required for Windows XP and Windows Server 2003.
ROOT/CIMV2	Win32_Process	Property	ExecutablePath	SeDebugPrivilege	The ExecutablePath property indicates the path to the executable file of the process.
		Property	MaximumWorkingSetSize	SeDebugPrivilege	The MaximumWorkingSetSize property indicates the maximum working set size of the process. The working set of a process is the set of memory pages currently visible to the process in physical RAM. These pages are resident and available for an application to use without triggering a page fault.
		Property	MinimumWorkingSetSize	SeDebugPrivilege	The MinimumWorkingSetSize property indicates the minimum working set size of the process. The working set of a process is the set of memory pages currently visible to the process in physical RAM. These pages are resident and available for an application to use without triggering a page fault.
		Method	Create	SeIncreaseQuotaPrivilege	The Create method creates a new process.
		Method	Terminate	SeDebugPrivilege	The Terminate method terminates a process and all of its threads.
ROOT/CIMV2	Win32_ComputerSystem	Property	SystemStartupDelay	SeSystemEnvironmentPrivilege	The SystemStartupDelay property indicates the time to delay before starting the operating system **Note:** The SE_SYSTEM_ENVIRONMENT privilege is required on IA64bit machines. This privilege is not required for 32-bit systems.
		Property	SystemStartupOptions	SeSystemEnvironmentPrivilege	The SystemStartupOptions property array indicates the options for starting up the computer system. Note that this property is not writable on IA64-bit machines. Constraints: Must have a value. **Note:** The SE_SYSTEM_ENVIRONMENT privilege is required on IA64-bit machines. This privilege is not required for other systems.
		Property	SystemStartupSetting	SeSystemEnvironmentPrivilege	The SystemStartupSetting property indicates the index of the default start profile. This value is 'calculated' so that it usually returns zero (0) because at write-time, the profile string is physically moved to the top of the list. (This is how Windows NT determines which value is the default.) **Note:** The SE_SYSTEM_ENVIRONMENT privilege is required on IA64bit machines. This privilege is not required for 32-bit systems.

Table 4.1 *The WMI Privileges Required for Some Classes, Properties, or Methods (continued)*

Namespace	WMI Class	CIM Element	Name	Privileges	Description
ROOT/CIMV2	Win32_OperatingSystem	Method	Reboot	SeShutdownPrivilege	The Reboot method shuts down the computer system, then restarts it. On computers running Windows NT/2000, the calling process must have the SE_SHUTDOWN_NAME privilege. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. Other - For integer values other than those listed above, refer to Win32 error code documentation.
		Method	Shutdown	SeShutdownPrivilege	The Shutdown method unloads programs and DLLs to the point where it is safe to turn off the computer. All file buffers are flushed to disk, and all running processes are stopped. On computer systems running Windows NT/2000, the calling process must have the SE_SHUTDOWN_NAME privilege. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. Other - For integer values other than those listed above, refer to Win32 error code documentation.
		Method	Win32Shutdown	SeShutdownPrivilege	The Win32Shutdown method provides the full set of shutdown options supported by Win32 operating systems. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. Other - For integer values other than those listed above, refer to Win32 error code documentation.
		Method	SetDateTime	SeSystemTimePrivilege	The SetDateTime method sets the current system time on the computer. On computer systems running Windows NT/2000, the calling process must have the SE_SYSTEMTIME_NAME privilege. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. Other - For integer values other than those listed above, refer to Win32 error code documentation.
ROOT/CIMV2	CIM_LogicalFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	CIM_DeviceFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	CIM_Directory	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	Win32_Directory	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	CIM_DataFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).

Table 4.1 *The WMI Privileges Required for Some Classes, Properties, or Methods (continued)*

Namespace	WMI Class	CIM Element	Name	Privileges	Description
ROOT/CIMV2	Win32_ShortcutFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	Win32_CodecFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	Win32_NTEventlogFile	Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
		Method	ClearEventlog	SeBackupPrivilege	Clears the specified event log, and optionally saves the current copy of the logfile to a backup file. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. 8 - The user does not have adequate privileges. 21 - Invalid parameter. Other - For integer values other than those listed above, refer to Win32 error code documentation.
		Method	BackupEventlog	SeBackupPrivilege	Saves the specified event log to a backup file. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. 8 - The user does not have adequate privileges. 21 - Invalid parameter. 183 - Archive file name already exists. Cannot create file. Other - For integer values other than those listed above, refer to Win32 error code documentation.
ROOT/CIMV2	Win32_PageFile	Class		SeCreatePagefilePrivilege	The Win32_PageFile class has been Deprecated in favor of the Win32_PageFileUsage and Win32_PageFileSetting. These classes respectively correspond to the runtime and persisted states of pagefiles. The Win32_PageFile represents the file used for handling virtual memory file swapping on a Win32 system. Note: The SE_CREATE_PAGEFILE privilege is required for Windows XP and Windows Server 2003.
		Method	GetEffectivePermission	SeSecurityPrivilege	The GetEffectivePermission method determines whether the caller has the aggregated permissions specified by the Permission argument not only on the file object, but on the share the file or directory resides on (if it is on a share).
ROOT/CIMV2	Win32_NTLogEvent	Class		SeSecurityPrivilege	This class is used to translate instances from the NT Eventlog.
ROOT/CIMV2	Win32_SecuritySetting	Method	GetSecurityDescriptor	SeRestorePrivilege	Retrieves a structural representation of the object's security descriptor
		Method	SetSecurityDescriptor	SeRestorePrivilege	Sets security descriptor to the specified structure
ROOT/CIMV2	Win32_LogicalFileSecuritySetting	Method	GetSecurityDescriptor	SeRestorePrivilege	Retrieves a structural representation of the object's security descriptor.
		Method	SetSecurityDescriptor	SeRestorePrivilege	Sets security descriptor to the specified structure.

Table 4.1 *The WMI Privileges Required for Some Classes, Properties, or Methods (continued)*

Namespace	WMI Class	CIM Element	Name	Privileges	Description
ROOT/CIMV2	Win32_LogicalShareSecuritySetting	Method	GetSecurityDescriptor	SeRestorePrivilege	Retrieves a structural representation of the object's security descriptor. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. 2 - The user does not have access to the requested information. 8 - Unknown failure. 9 - The user does not have adequate privileges. 21 - The specified parameter is invalid. Other - For integer values other than those listed above, refer to Win32 error code documentation.
		Method	SetSecurityDescriptor	SeRestorePrivilege	Sets security descriptor to the specified structure. The method returns an integer value that can be interpretted as follows: 0 - Successful completion. 2 - The user does not have access to the requested information. 8 - Unknown failure. 9 - The user does not have adequate privileges. 21 - The specified parameter is invalid. Other - For integer values other than those listed above, refer to Win32 error code documentation.
ROOT/CIMV2	Win32_PageFileSetting	Class		SeCreatePagefilePrivilege	The Win32_PageFileSetting class represents the settings of a page file. Information contained within objects instantiated from this class specify the page file parameters used when the file is created at system startup. The properties in this class can be modified and deferred until startup. These settings are different from the run time state of a page file expressed through the associated class Win32_PageFileUsage. **Note:** The SE_CREATE_PAGEFILE privilege is required for Windows XP and Windows Server 2003.
ROOT/CIMV2	Win32_NTLogEventLog	Class		SeSecurityPrivilege	The Win32_NTLogEventLog class represents an association between an NT log event and the log file that contains the event.
ROOT/CIMV2	CIM_ProcessExecutable	Class		SeDebugPrivilege	A link between a process and a data file indicating that the file participates in the execution of the process. **Note:** In order to receive all possible instances of this class, the SE_DEBUG_PRIVILEGE should be enabled.
ROOT/CIMV2	Win32_NTLogEventUser	Class		SeSecurityPrivilege	The Win32_NTLogEventUser class represents an association between an NT log event and the active user at the time the event was logged.
ROOT/CIMV2	Win32_NTLogEventComputer	Class		SeSecurityPrivilege	The Win32_NTLogEventComputer class represents an association between an NT log event and the computer from which the event was generated.

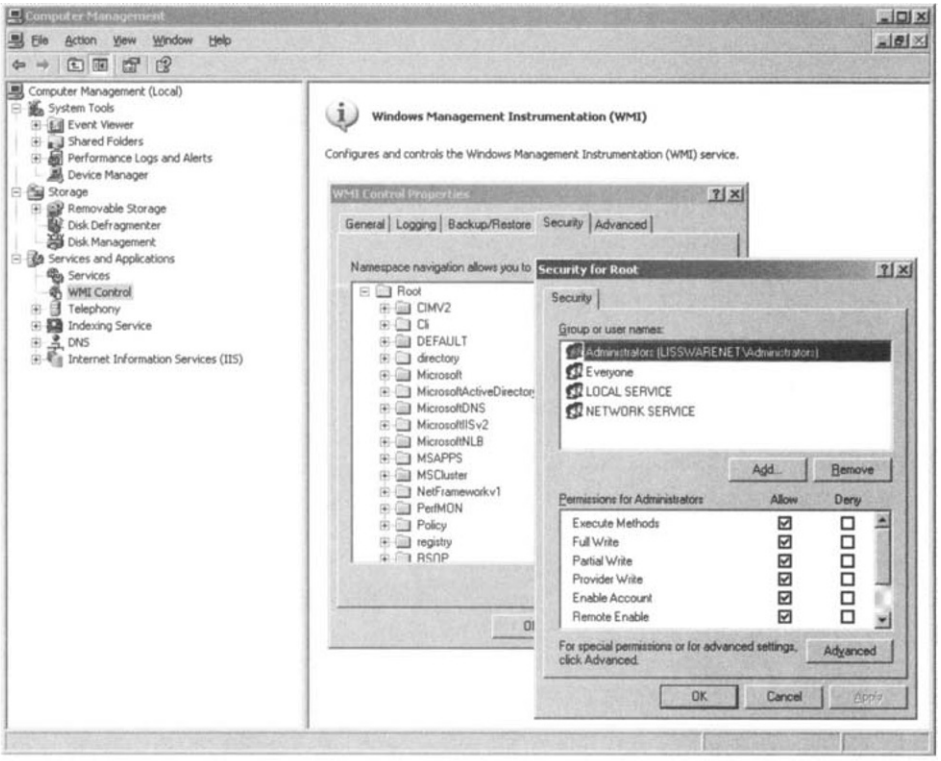


Figure 4.1 *The default security settings on the Root namespace.*

Table 4.2 *The Default Right Settings on the WMI Root Namespace*

	Administrators	Everyone	Local Service	Network Service
Execute Methods	X	X	X	X
Full Write	X			
Partial Write	X			
Provider Write	X	X	X	X
Enable Account	X	X	X	X
Remote Enable	X			
Read Security	X			
Edit Security	X			

4.3 WMI and Active Server Page (ASP)

Most of the scripting techniques we learned in previous chapters are applicable to Active Server Page (ASP) scripts. However, to run a WMI script from an Active Server Page (ASP) script, it is important to note two important points, where security is the main aspect to consider when developing WMI-ASP solutions. This is why we cover this information in this chapter. The two points to remember are:

1. The asynchronous scripting technique cannot be used within an ASP script, because the VBScript *CreateObject* statement does not support the connection to a sink routine, such as the *CreateObject* statement used with WSH (i.e., *Wscript.CreateObject* statement).
2. Regarding the security settings: Some parameters must be carefully considered regarding the platform the ASP script is run from.

4.3.1 Authentication settings

Authentication settings are as follows:

- **Security configuration under Windows NT 4.0:** To run a WMI-ASP script under the Windows NT 4.0 platform, a registry key must be modified to grant the Scripting API all of the permissions of the account running Internet Information Services (IIS). If not, then the browser client must contain the necessary permissions and security settings. The registry key to be modified is located in the *HKLM\Software\Microsoft\WBEM\Scripting* registry hive, where the DWORD value *Enable for ASP* must be set to 1 (see Figure 4.2). Of

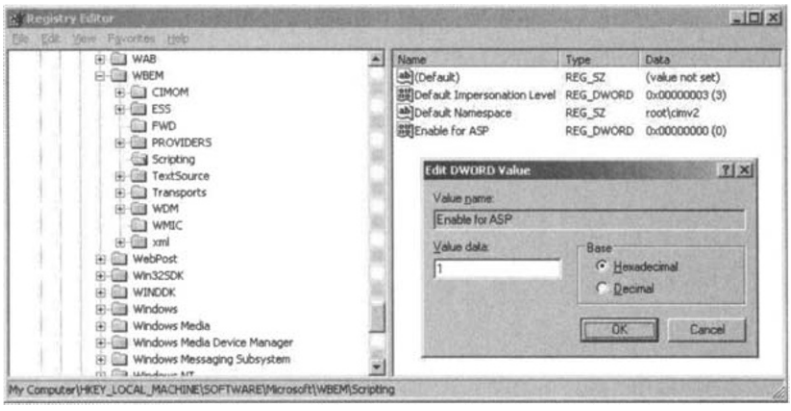


Figure 4.2 *Granting all Scripting API permissions to the IIS account.*

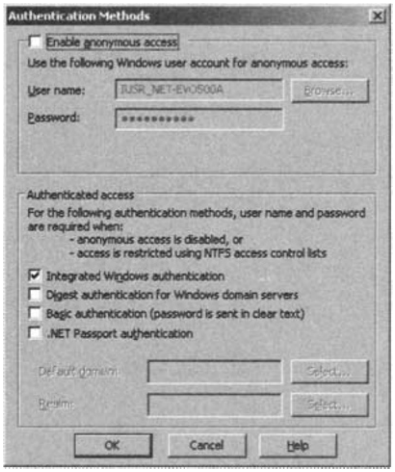


Figure 4.3 *Setting the Windows Integrated Authentication.*

course, changing this registry key is granting all privileges to the account running IIS, which represents some danger, since you will be granting all privileges without any further considerations. We will see later in this section that different approaches can be used to minimize such a risk.

- **Security configuration under Windows 2000 and above:** Under Windows 2000 and above, there are several ways to customize the IIS security. Of course, since a WMI-ASP script can perform some critical tasks, it is important to request the user to authenticate. Therefore, the recommended approach is to disable the anonymous access for the authentication protocol used. As we will see, the authentication method used will impact the WMI security configuration. Under Windows 2000 and above, you have a choice:
  - **The Windows Integrated Authentication:** As we can see in Figure 4.3, the Windows Integrated Authentication (WIA) is enabled for ASP, while the anonymous access is disabled. This security setting doesn't require any specific WMI security configuration. Enabling the Windows Integrated Authentication implies the use of Kerberos or NTLM.
  - **Passport or digest authentication:** The passport or digest authentications (see Figure 4.4) require that the CIM repository namespace *Remote Enable* privilege be granted to accounts authenticated by one of these protocols, since the user access is treated as a remote user access. For instance, creating a Windows Security Global Group and including the users authorized to remote access

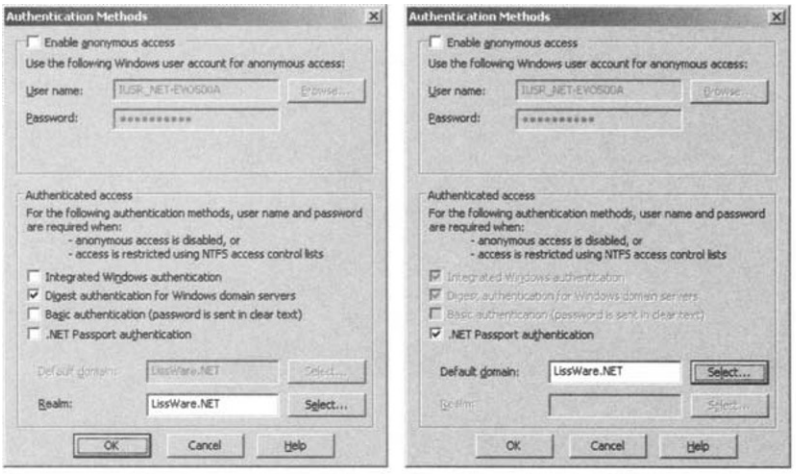


Figure 4.4
Setting the passport or digest authentication.

a selected namespace is a good practice (see Figure 4.5). Note that with the digest authentication, passwords are still wrapped in a password encryption key, but they are not hashed.

- **Anonymous and basic authentications:** For obvious security reasons, it is recommended not using the anonymous and basic authentications, which are a clear-text authentication without encryption. Obviously, anonymous basically means "no authentication," which is not a recommended approach. However, the *Remote Enable* privilege is not required for these two authentication methods if they are used.

Figure 4.5
Enabling remote access for remote users.

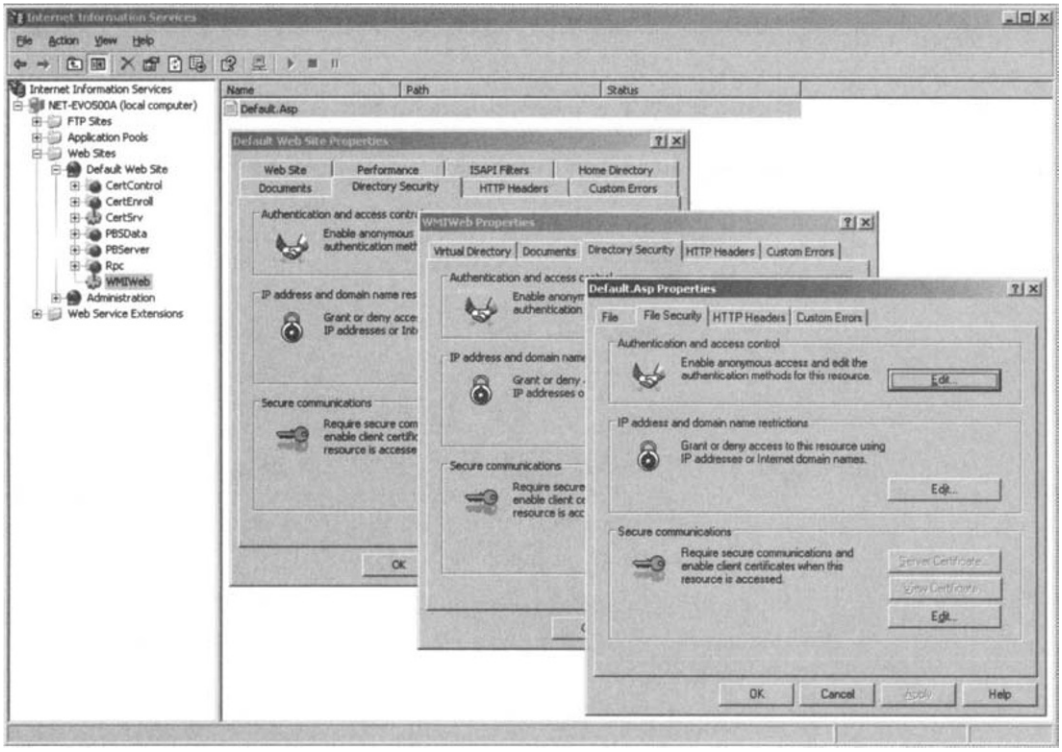


Figure 4.6 *The three IIS locations where authentication can be defined.*

4.3.2 Customizing IIS 5.0 and above

Regarding the security configuration under Internet Information Server (IIS) 5.0 and above, the authentication settings previously reviewed are not the only things to consider when setting up an IIS server running WMI-ASP scripts. Actually, the location of the authentication-level definitions is also very important. IIS can enforce the authentication at the Web Server level, the virtual directory, or the file level (see Figure 4.6).

Moreover, for WMI-dedicated security, it is a good practice to create an independent directory structure, which contains all HTML pages and WMI-ASP scripts (see Figure 4.7). This organization allows you to customize specific security settings independently from the rest of the server security. For example, Figure 4.7 shows that the *WMI-ASP Script* folder is not under the *WWWRoot* folder, which allows the creation of specific access rights on the file system to access the WMI-ASP scripts.

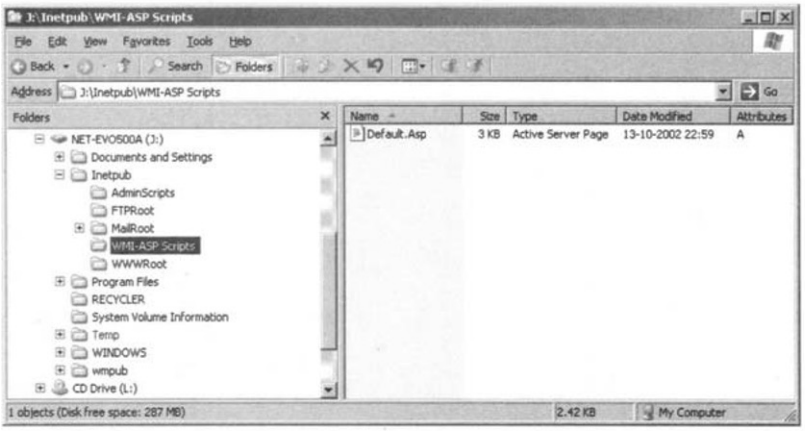


Figure 4.7
The isolated file system directory from the WWWRoot folder.

In order to increase the WMI-ASP script security, it is a good idea to disable the anonymous access and enable the Windows Integrated Authentication (WIA), as shown in Figure 4.3.

However, there are some situations where anonymous access is required. By default, Web clients accessing IIS are using the IIS logon identity (i.e., IUSR_<machine_name>). Therefore, it is a good idea for a more secure approach to create a new interactive user account. This allows you to define security settings that suit the *WMI-ASP Scripts* virtual Web site requirements without impacting the overall IIS security (unless you modify the security at the server level). From the directory security tab of the virtual directory properties, you can define the new logon identifier and enter the

Figure 4.8
Enabling anonymous access with IIS.

Figure 4.9 *Ensuring WMI CIM repository access for the WMI-ASP dedicated account.*

appropriate password in the authentication methods user interface, as shown in Figure 4.8.

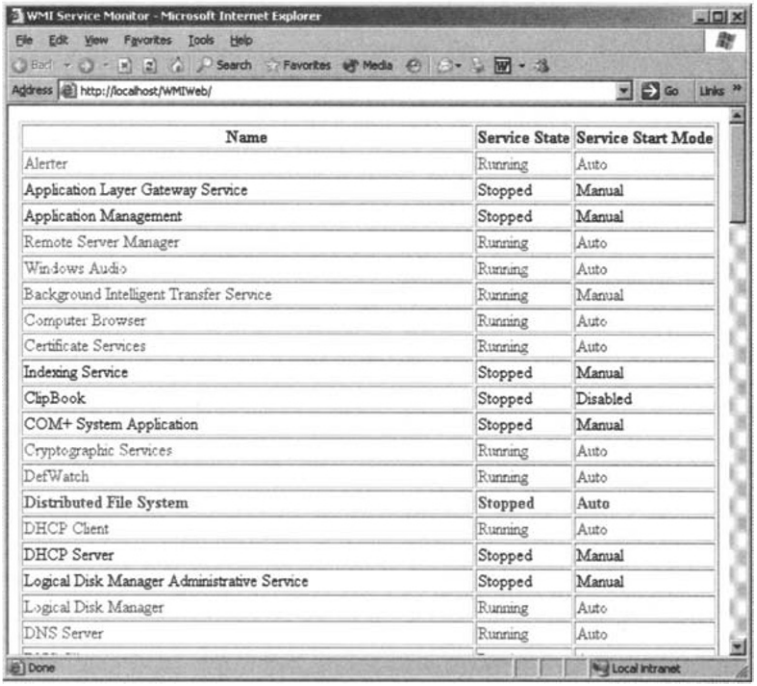
Note that to secure applications, IIS 5.0 also has the ability to isolate applications in different security contexts. IIS 6.0 uses the concept of application pools to implement the same functionality. You can refer to the IIS documentation for more information about these security features.

Of course, since a different user account for the virtual Web site identity is used, you must make sure that this user account has access to the CIM repository accessed by the WMI-ASP script. For instance, if the WMI-ASP script uses the *Win32_Service* class located in the **Root\CIMv2** namespace, the *WMI-ASP-Script* user must be enabled and have remote access granted on that namespace. Even if everything is executed locally, the remote access must be granted, since the user account is treated by IIS as a remote user access (see Figure 4.9).

From a scripting point of view, developing WMI logics in ASP pages is the same as developing WMI logics in WSH (the exception, as mentioned in the beginning of this section, is that asynchronous calls are not supported). Of course, since the scripting environment is different, the output of information must be handled in respect to the ASP context and must make use of HTML tags.

Sample 4.1 shows an example of an ASP script listing the state of all Windows services in an HTML page (see Figure 4.10). To get this ASP script running, make sure that the IIS security settings are set accordingly, as explained previously, in regard to the operating system and IIS version used.

Figure 4.10 *Viewing all Win32_Service instance states from the Web.*



Name	Service State	Service Start Mode
Alerter	Running	Auto
Application Layer Gateway Service	Stopped	Manual
Application Management	Stopped	Manual
Remote Server Manager	Running	Auto
Windows Audio	Running	Auto
Background Intelligent Transfer Service	Running	Manual
Computer Browser	Running	Auto
Certificate Services	Running	Auto
Indexing Service	Stopped	Manual
ClipBook	Stopped	Disabled
COM+ System Application	Stopped	Manual
Cryptographic Services	Running	Auto
DefWatch	Running	Auto
Distributed File System	Stopped	Auto
DHCP Client	Running	Auto
DHCP Server	Stopped	Manual
Logical Disk Manager Administrative Service	Stopped	Manual
Logical Disk Manager	Running	Auto
DNS Server	Running	Auto

Sample 4.1 *Viewing all Win32_Service instances with their status from a WMI-ASP script*

```
 1:<%@ LANGUAGE="VBSCRIPT"%>
 2:
 3:<html>
 4: <head>
 5:  <title>WMI Service Monitor</title>
 6: </head>
 7:
 8: <body>
 9: <%
10:   Dim objWMIServices
11:   Dim objWMIInstance, objWMIInstances
12:
13:   Set objWMILocator = CreateObject("WbemScripting.SWbemLocator")
14:   Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace)
15:
16:   If Err.Number = 0 Then
17:      Set objWMIInstances = objWMIServices.InstancesOf ("Win32_Service")
18: %>
19:      <table BORDER="1">
20:      <tr>
21:       <th>Name</th>
22:       <th>Service State</th>
23:       <th>Service Start Mode</th>
24:      </tr>
25: <%
26:      For Each objWMIInstance in objWMIInstances
```

```
27:          If objWMIInstance.Started = False And objWMIInstance.StartMode = "Auto" Then
28: %>
29:             <tr>
30:              <td>
31:               <font color="#FF0000"><b>
32:               <%=objWMIInstance.DisplayName%>
33:               </b></font>
34:              </td>
35:              <td>
36:               <font color="#FF0000"><b>
37:               <%=objWMIInstance.State%>
38:               </b></font>
39:              </td>
40:              <td>
41:               <font color="#FF0000"><b>
42:               <%=objWMIInstance.StartMode%>
43:               </b></font>
44:              </td>
45:             </tr>
46: <%
47:          Else
48:             If objWMIInstance.Started = True Then
49: %>
50:             <tr>
51:              <td>
52:               <font color="#008000">
53:               <%=objWMIInstance.DisplayName%>
54:               </font>
55:              </td>
56:              <td>
57:               <font color="#008000">
58:               <%=objWMIInstance.State%>
59:               </font>
60:              </td>
61:              <td>
62:               <font color="#008000">
63:               <%=objWMIInstance.StartMode%>
64:               </font>
65:              </td>
66:             </tr>
67: <%
68:             Else
69: %>
70:             <tr>
71:              <td>
72:               <%=objWMIInstance.DisplayName%>
73:              </td>
74:              <td>
75:               <%=objWMIInstance.State%>
76:              </td>
77:              <td>
78:               <%=objWMIInstance.StartMode%>
79:              </td>
80:             </tr>
81: <%
82:             End If
83:          End If
84:      Next
85: %>
86:      </table>
```

```
87: <%
88:   Else
89: %>
90:       <table>
91:        <tr>
92:         <td>Error - <%=Err.description%>, <%=Err.number%>, <%=Err.Source%></td>
93:        </tr>
94:       </table>
95: <%
96:   End If
97:  %>
98: </body>
99:</html>
```

4.4 WMI security descriptor management

4.4.1 The security descriptor WMI representation

Under Windows, for the purposes of scripting, the security descriptor structure is abstracted by a collection of COM objects. For instance, the Active Directory Service Interfaces (ADSI) have their own collection of COM objects to represent a security descriptor. In the same way, WMI also has its own way to represent the security descriptor structure. In this particular case, WMI uses three classes to represent the security descriptor:

- *Win32_SecurityDescriptor*
- *Win32_ACE*
- *Win32_Trustee*

Note that security descriptors are not dedicated to Windows. Each operating system has its own way of structuring the content of a security descriptor and its own interfaces for manipulating the security descriptor content.

Before looking at classes exposing security information, we will first take a quick look at the security descriptor logical structure, as seen from WMI (see Figure 4.11). This will help us to understand the class purposes.

The security descriptor is made up of several components, which expose properties, which in turn may contain subcomponents. From a WMI point of view, the security descriptor is an **SWBemObject** object, which is an instance of the *Win32_SecurityDescriptor* class. Basically, a security descriptor as represented by WMI is made up of four properties, which contain other objects:

- **Owner:** It identifies the owner of the security descriptor. The owner is represented by an instance of the *Win32_Trustee* class.

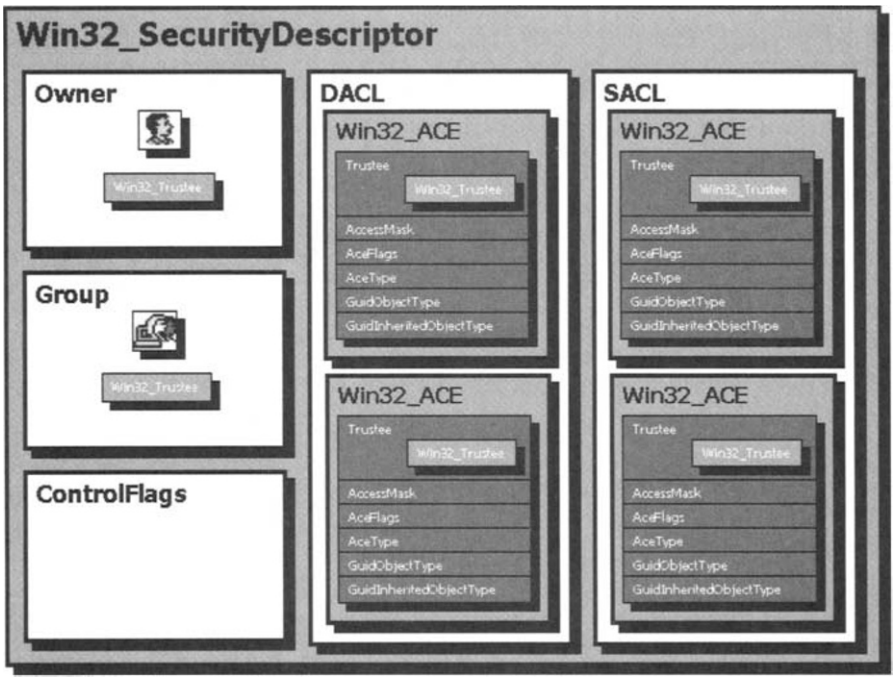


Figure 4.11 *The security descriptor logical structure as seen from WMI.*

- **Group:** It identifies the primary group of the security descriptor. The group is represented by an instance of the *Win32_Trustee* class. This property contains the SID for the owner's primary group. This information is only used by the POSIX subsystem of Windows and is ignored by other Windows subsystems (i.e., Win32). Therefore, the Group property is only present for compatibility purposes regarding the POSIX applications.
- **Control:** It is a set of bit flags that describes a security descriptor and/or its components. It is mainly used to determine the presence of the Discretionary ACL (DACL) and the System ACL (SACL) and the effect of the inherited Access Control Entries (ACE) on the security descriptor. We will see in section 4.11.2 ("Deciphering the security descriptor *Control Flags*") how to decipher this value.
- **ACL:** The Access Control List (ACL) is an array holding Access Control Entries (ACE). An ACE object is represented by an instance of the *Win32_ACE* class, which defines, on a per user basis, what the user is authorized to do. This is the smallest data element defining the user privileges and permissions of an object.

There are two kinds of ACLs: the Discretionary ACL (DACL) and the System ACL (SACL). The DACL handles access control on objects. The SACL handles the system auditing on objects. To define each privilege and permission for a given user, an ACE has several properties. An ACE is composed of:

- **Trustee:** A trustee is a user, a group, or a computer with access rights to an object (i.e., DomainName\UserName). As for the owner and the group element of the security descriptor, a trustee is represented by an instance of the *Win32_Trustee* class.
- **AccessMask:** The access mask is a sequence of bits turned ON or OFF to activate specific rights. Note that the bit values are not the same for all securable objects. For instance, the bit sequence defining rights on a file is different from the bit sequence defining rights on an Active Directory object. In section 4.11.4.5 ("Deciphering the *ACE AccessMask* property"), we will see which bit sequence to use in regard to the secured object.
- **AceFlags:** This entry contains the inheritance control flags. As for the AccessMask, the bit sequence defining the inheritance may vary from one secured object to another. In section 4.11.4.3 ("Deciphering the *ACE Flags* property"), we will see which bit sequence to use in regard to the managed object.
- **AceType:** The value assigned to this ACE property determines the type of permissions granted for the corresponding access mask (Allowed, Denied, Audit).
- **GuidObjectType:** This property indicates what object class or Active Directory attribute set an ACE refers to. It takes a GUID as a value. Usually, this property is set when referring to an Active Directory Extended Right or to an Active Directory class.
- **GuidInheritedObjectType:** This property specifies the GUID of an object class whose instances will inherit the ACE.

4.4.2 How to access the security descriptor

The structural view of the security descriptor is the same for any object in a Windows system (i.e., share or file system objects such as files and folders). However, not all WMI classes representing manageable entities provide an access method to retrieve the security descriptor. The WMI provider capabilities related to a manageable entity and the classes it supports are deter-

minant. For instance, as we will see further in section 4.7.1.1 ("Retrieving file and folder security descriptors with WMI"), the *Win32_LogicalFileSecuritySetting* class, supported by the WMI *Security* provider, exposes the *GetSecurityDescriptor* method to retrieve the security descriptor set on a file or a folder.

Unfortunately, it is not always easy or possible to retrieve a security descriptor from a secured object with WMI. In some cases, a more suitable COM abstraction layer such as ADSI must be used. When doing so, it is important to consider the secured object type to be managed when choosing an access method. For instance, the technique to access a security descriptor on a file or a folder will not necessarily be the same as the technique used to access the security descriptor on a registry key. Here, we totally rely on the COM abstraction layer capabilities (i.e., WMI versus ADSI).

Actually, we must clearly distinguish two things:

- **The technique for accessing and updating the security descriptor:** This relies on the WMI provider and class capabilities. If there is no WMI class method or properties exposing the desired security descriptor, it is possible that another technique not implemented by WMI must be used. In such a case, if ADSI offers some capabilities, we will use it to retrieve the security descriptor from a secured object.
- **The format of the security descriptor:** Once the security descriptor is read, its representation is not always in the form of a *Win32_SecurityDescriptor* instance. From a WMI perspective, some methods or properties return the security descriptor as a *Win32_SecurityDescriptor* instance; others return the security descriptor in a raw form (a binary array). If the security descriptor is accessed with ADSI, then its representation will be available in the ADSI object model, which is totally different from the *Win32_SecurityDescriptor* class representation supported by WMI.

4.4.3 The security descriptor ADSI representation

Because we will require help from ADSI, it is important to have a look at the security descriptor representation made by ADSI (Figure 4.12).

Although the security descriptor structure remains globally the same as the one shown in Figure 4.11, only the names of the components representing a security descriptor with their properties are slightly different.

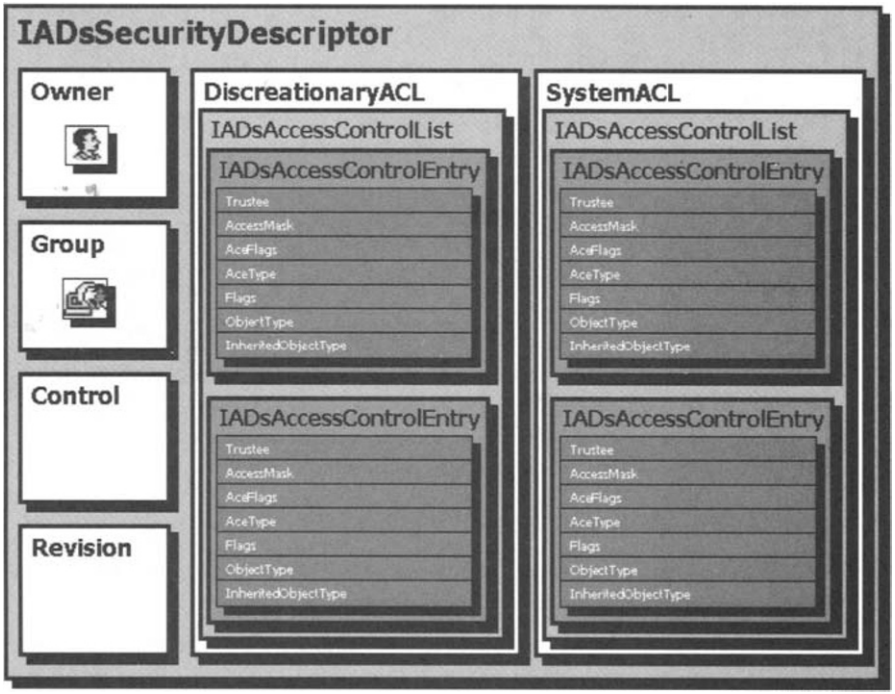


Figure 4.12 *The security descriptor logical structure as seen from ADSI.*

However, the information they contain is identical. Table 4.3 presents a short comparison of the properties exposed by both object models.

As we can see, WMI uses three classes to represent a security descriptor and its components (*Win32_SecurityDescriptor*, *Win32_ACE*, and *Win32_Trustee*). We clearly see in Figure 4.11 and Table 4.3 that WMI does not represent an ACL as an object. WMI represents an ACL as an array containing a series of ACEs exposed by the DACL or SACL properties. On the other hand, ADSI exposes the same information type, but it is organized differently. First, a **SecurityDescriptor** object (exposed by the **IADsSecurityDescriptor** ADSI COM interface) represents a security descriptor. Next, the *DiscretionaryACL* or the *SystemACL* properties expose the ACL in an **AccessControlList** object (exposed by the **IADsAccessControlList** ADSI COM interface), which, in turn, contains a collection of ACEs represented by **AccessControlEntry** objects (exposed by the **IADsAccessControlEntry** ADSI COM interface).

The only extra information we get from ADSI is the revision level of the security descriptor and some flags that are turned ON when values are set in

Table 4.3 *The WMI and ADSI Security Descriptor Exposed Methods and Properties*

WMI		ADSI	
Win32_SecurityDescriptor		**IADsSecurityDescriptor**	
Properties		**Properties**	
Owner	object:Win32_Trustee	Owner	string
Group	object:Win32_Trustee	Group	string
ControlFlags	unit32	Control	long
DACL	array of object:Win32_ACE	DiscretionaryACL	object:IADsAccessControlList
SACL	array of object:Win32_ACE	SystemACL	object:IADsAccessControlList
N/A		Revision	long
Methods		**Methods**	
N/A		CopySecurityDescriptor()	
		IADsAccessControlList	
Properties		**Properties**	
N/A		AclRevision	long
		AceCount	long
Methods		**Methods**	
N/A		AddAce (objACE)	
		RemoveAce (objACE)	
		CopyAccessList()	
		(Enumeration)	collection of object:IADsAccessControlEntry
Win32_ACE		**IADsAccessControlEntry**	
Properties		**Properties**	
Trustee	object:Win32_Trustee	Trustee	string
AccessMask	unit32	AccessMask	long
AceFlags	unit32	AceFlags	long
AceType	unit32	AceType	long
N/A		Flags	long
GUIDObjectType	string	ObjectType	string
GUIDInheritedObjectType	string	InheritedObjectType	string
Methods		**Methods**	
N/A		N/A	

the *ACE ObjectType* and *ACE InheritedObjectType* properties. In sections 4.11.4.5.3.1 ("Understanding the ACE ObjectType property") and 4.11.4.5.3.2 ("Understanding the ACE InheritedObjectType property"), we will see how to use these properties when working with Active Directory security descriptors.

4.4.4 Which access technique to use? Which security descriptor representation do we obtain?

As mentioned previously, based on the secured object type we access, it is important to determine the access methods available and the security descriptor format associated with the access method. Table 4.4 helps us understand which access method can be used and which security descriptor representation format is obtained, based on the security descriptor origin.

It is interesting to note that sometimes it is possible to use both WMI and ADSI to retrieve a security descriptor. For instance, this is the case for a security descriptor from a file or a folder. In some other cases, only WMI or ADSI can be used. This is the case for a security descriptor from a registry key, which only uses an ADSI access method. It is the same for a security

Table 4.4 *The Security Descriptor Access Methods with their Representations*

		File		Folder		Share		AD object		E2K mailbox			Registry key		WMI namespace	
		ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	CDOEXM	WMI	ADSI	WMI	ADSI	WMI
Source	READ	FS	FS	FS	FS	FS	FS	AD	AD	AD	E2K Store	AD	REG	-	-	CIM
SD retrieval technique	READ	ADSI[1]	WMI	ADSI[1]	WMI	ADSI[3]	WMI	ADSI	WMI	ADSI	CDOEXM[8]	WMI	ADSI[1]	-	-	WMI
SD format	READ	ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	Raw	ADSI	ADSI	Raw	ADSI	-	-	Raw
SD conversion (raw->)	READ	-	-	-	-	-	-	-	ADSI[5]	-	-	ADSI[5]	-	-	-	ADSI[5]
SD object model	READ	ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	ADSI	ADSI	ADSI	ADSI	ADSI	-	-	ADSI
Owner available?	READ	Y	Y	Y	Y	-	-	Y	Y	Y	Y	Y	Y	-	-	Y
Group available?	READ	Y	Y	Y	Y	-	-	Y	Y	Y	Y	Y	Y	-	-	Y
Security Descriptor revision?	READ	Y	N	Y	N	Y	N	Y	-	Y	Y	-	Y	-	-	-
DiscretionaryACL available?	READ	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y
SystemACL available?	READ	Y[2]	Y	Y[2]	Y	-	-	Y[4]	N[6]	Y[4]	Y	N[6]	Y[2]	-	-	-
SD object model	WRITE	ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	ADSI	ADSI	ADSI	ADSI	ADSI	-	-	ADSI
SD conversion (-> raw)	WRITE	-	-	-	-	-	-	-	ADSI[5]	-	-	ADSI[5]	-	-	-	ADSI[5]
SD format	WRITE	ADSI	WMI	ADSI	WMI	ADSI	WMI	ADSI	Raw	ADSI	ADSI	Raw	ADSI	-	-	Raw
SD update technique	WRITE	ADSI[1]	WMI	ADSI[1]	WMI	ADSI[3]	WMI	ADSI	WMI	ADSI	CDOEXM[8]	WMI	ADSI[1]	-	-	WMI
Target	WRITE	FS	FS	FS	FS	FS	FS	AD	AD	AD[7]	E2K Store	AD[7]	REG	-	-	CIM

(1) **Windows NT 4.0/Windows 2000:** Requires **ADsSecurity.DLL** from the ADSI Resource Kit
Windows XP/Windows Server 2003: Requires ADSI **ADsSecurityUtility** object.

(2) **Windows NT 4.0/Windows 2000:** SystemACL is not available.
Windows XP/Windows Server 2003: By design SystemACL is only available with the **ADsSecurityUtility** object with statement:

```
objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
                              ADS_SECURITY_INFO_GROUP Or _
                              ADS_SECURITY_INFO_DACL Or _
                              ADS_SECURITY_INFO_SACL
```

Warning: Unfortunately, a bug in the **ADsSecurityUtility** object prevents the retrieval of the SystemACL. Microsoft doesn't plan to fix this bug in the RTM code for timing issues (WMI offers an acceptable work-around for file and folders only. For the registry key, there is no work-around available).

(3) **Windows XP/Windows Server 2003:** Requires ADSI **ADsSecurityUtility** object. Not supported under Windows 2000.

(4) **Windows 2000 and Windows XP/Windows Server 2003:** Requires statement:

```
objADObject.SetOption ADS_OPTION_SECURITY_MASK, ADS_SECURITY_INFO_OWNER Or _
                      ADS_SECURITY_INFO_GROUP Or _
                      ADS_SECURITY_INFO_DACL Or _
                      ADS_SECURITY_INFO_SACL
```

(5) **Windows XP/Windows Server 2003:** Security descriptor conversion can done with **ADsSecurityUtility** object or ADSI legacy interfaces.
Windows NT 4.0/Windows 2000: Security descriptor conversion can done with ADSI legacy interfaces.

(6) **Windows NT 4.0/Windows 2000/Windows XP/Windows Server 2003:** SystemACL is not available.

(7) Only if the mailbox is NOT yet created in the Exchange 2000 Web Store.

(8) Requires Exchange 2000 Service Pack 2 (or above) and Exchange System Manager installation for CDOEXM installation.

descriptor from a CIM repository namespace, which only uses a WMI access method. There is one particular case concerning the Exchange 2000 mailbox security descriptor. We see that we can get the security descriptor by using ADSI, WMI, or Collaboration Data Objects for Exchange Management (CDOEXM). Of course, each technique has its own peculiarities, which must be considered. We will examine this in section 4.7 ("Accessing the security descriptor set on manageable entities").

However, not all access methods retrieve security descriptors in the same format. It makes sense to think that an ADSI access method will retrieve the security descriptor in the ADSI object model and a WMI access method will retrieve the security descriptor in a *Win32_SecurityDescriptor* instance. However, there are exceptions. For instance, it is possible to retrieve a security descriptor from an Active Directory object with ADSI or with WMI. The ADSI access method will expose the security descriptor in the ADSI object model, while the WMI access method retrieves the security descriptor in binary form. In the latter case, the security descriptor needs to be converted to a structural representation. The conversion of a binary array representing a security descriptor to a *Win32_SecurityDescriptor* instance is not simple. This may require some advanced programming techniques on top of the Win32 API to decipher the security descriptor. Obviously, we leave the world of scripting to enter the world of API programming. Hopefully, with ADSI, it is possible to convert the binary array to a **SecurityDescriptor** object. We will see in section 4.9 ("The security descriptor conversion") how to proceed. This means that in some cases, we can access the security descriptor with WMI and manipulate its content with the ADSI object model. Although slightly more complex, this method works well and keeps programming in the space of the scripting world.

Last but not least, not all access methods retrieve all security descriptor properties. As seen before, ADSI shows the revision level of the security descriptor, while WMI does not. Most importantly, the System ACL is also subject to some exceptions. As we can see in Table 4.4, based on the security descriptor origin (file, share, Active Directory, etc.) and the access method used to read the security descriptor, the System ACL is not always available. For instance, this is the case for an Active Directory object retrieved with WMI, where the SystemACL is missing its ADSI representation. The Operating System platform also influences this, since the File System share security descriptor ADSI access method is not available under Windows 2000 or any earlier platforms.

4.5 The WMI Security provider

Since WMI implements some security scripting capabilities, it is interesting to examine closely the WMI *Security* provider capabilities. Available under Windows NT, Windows 2000, Windows XP, and Windows Server 2003, and registered in the **Root\CIMv2** namespace, the WMI *Security* provider is implemented as an instance and a method provider. It allows you to

Table 4.5 *The Security Provider Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
WMI security Provider																
SECRCW32	Root/CIMV2		X	X				X	X	X	X	X	X	X	X	X

retrieve or change security settings that control ownership, auditing, and access rights to the files, directories, and shares. Table 4.5 summarizes the WMI *Security* provider capabilities.

The WMI *Security* provider supports several WMI classes representing the security settings on files, folders, and shares. These classes are listed in Table 4.6.

Since the WMI *Security* provider retrieves security settings from the file system, it is clear that an association exists between the CIM representation of the file system object (i.e., file, directory, or share) and its security settings CIM representation. We can see this association in Figure 4.13A, where the *CIM_LogicalFile* class is associated with the *Win32_LogicalFileSecuritySetting* class by the *Win32_SecuritySettingOfLogicalFile* Association class. Remember that the *CIM_LogicalFile* is a superclass for the *CIM_DataFile* and *CIM_Directory* classes (see Chapter 2, Figure 2.12, "The *CIM_LogicalFile* class and its child classes." It is interesting to note

Table 4.6 *The Security Providers Classes*

Name	Type	Comments
Win32_Ace	Dynamic	Specifies an access control entry (ACE). An ACE grants permission to execute a restricted operation, such as writing to a file or formatting a disk. ACEs particular to WMI allow logons, remote access, method execution, and writing to the WMI repository.
Win32_LogicalFileSecuritySetting	Dynamic	Represents security settings for a logical file.
Win32_LogicalShareSecuritySetting	Dynamic	Represents security settings for a logical file.
Win32_SecurityDescriptor	Dynamic	Represents a security descriptor structure.
Win32_SID	Dynamic	Represents an arbitrary security identifier (SID). This property cannot be enumerated.
Win32_Trustee	Dynamic	Specifies a trustee. Either a name or a SID (byte array) can be used.
Win32_AccountSID	Association	Relates a security account instance with a security descriptor instance.
Win32_LogicalFileAccess	Association	Relates the security settings of a file/directory and one member of its DACL.
Win32_LogicalFileAuditing	Association	Relates the security settings of a file/directory and one member of its SACL.
Win32_LogicalFileGroup	Association	Relates the security settings of a file/directory and its group.
Win32_LogicalFileOwner	Association	Relates the security settings of a file/directory and its owner.
Win32_LogicalShareAccess	Association	Relates the security settings of a share and one member of its DACL.
Win32_LogicalShareAuditing	Association	Relates the security settings of a share and one member of its SACL.
Win32_SecuritySettingOfLogicalFile	Association	Represents security settings of a file or directory object.
Win32_SecuritySettingOfLogicalShare	Association	Relates the security settings of a share object with the object.

that the *Win32_LogicalFileSecuritySetting* class represents the exact same file system object as the *CIM_LogicalFile* superclass and that both classes use the same key property to locate their corresponding instances, which is the path of the file or the folder they represent. However, the set of properties exposed by both classes is different. Basically, the *CIM_LogicalFile* class exposes information purely related to the file system settings (i.e., creation date, size, attribute settings, etc.), while the *Win32_LogicalFileSecuritySetting* class represents some security settings (control flags, owner permissions). You can use **WMI CIM Studio** or the **LoadCIMInXL.Wsf** script shown in Sample 4.32 in the appendix to examine and compare the properties exposed by both classes.

As mentioned before (see Figure 4.11), a security descriptor contains a Discretionary ACL (to define access settings) and a System ACL (to define auditing settings). Therefore, a file system object has relationships with the *Win32_SID* class representing the granted entities at the Discretionary ACL and System ACL levels. These relationships are represented in Figures 4.13B and 4.13C.

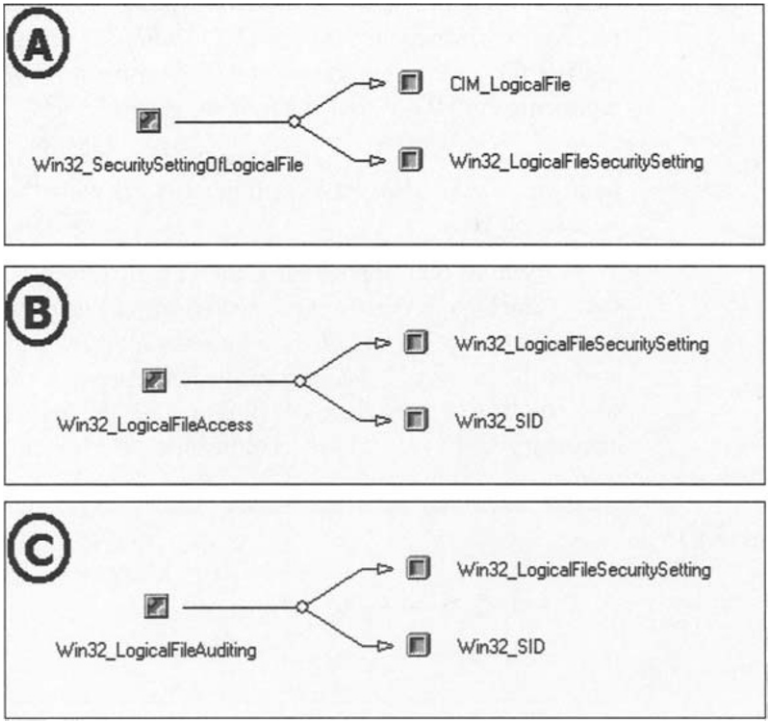


Figure 4.13 *The Win32_LogicalFileSecuritySetting class associations.*

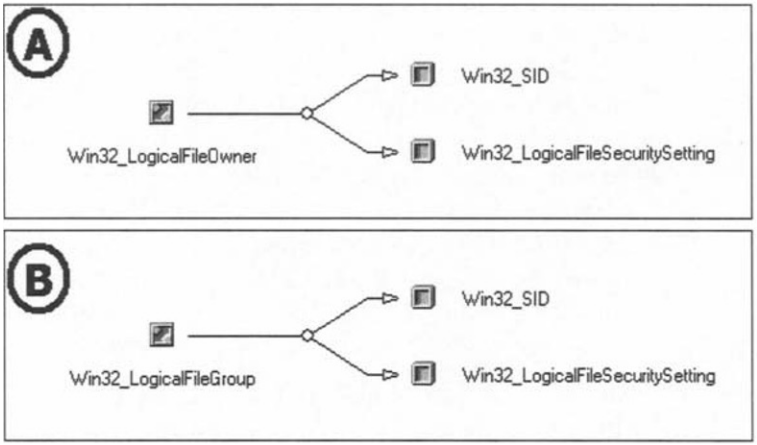


Figure 4.14 *The owner and group associations.*

We also mentioned that a security descriptor contains an owner and a group. This implies that a file system object also has relationships with the WMI classes representing the owner and the group coming from the security descriptor. These relationships are represented in Figure 4.14A for the owner and Figure 4.14B for the group. Since a user and a group correspond to a security principal, the *Win32_LogicalFileSecuritySetting* class representing the file system object is associated with the *Win32_SID* class, which represents the SID of the owner or the group.

To resolve the SID to an account, the *Win32_SID* class is associated with the *Win32_Account* class (Figure 4.15) with the *Win32_AccountSID* association class.

Everything that applies for a file or a directory is also applicable for a share. The logic is exactly the same. However, since a share doesn't have any owner or group, the *Win32_LogicalShareSecuritySetting* class is only associated with the *Win32_Share* class, which represents the share instance, and with the *Win32_SID* class, which represents the granted entities at the Discretionary ACL level. These relationships are represented in Figure 4.16.

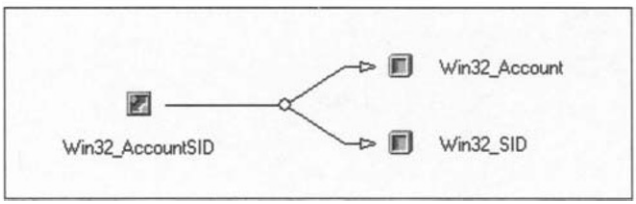


Figure 4.15 *The Win32_Account class and the Win32_SID class association.*

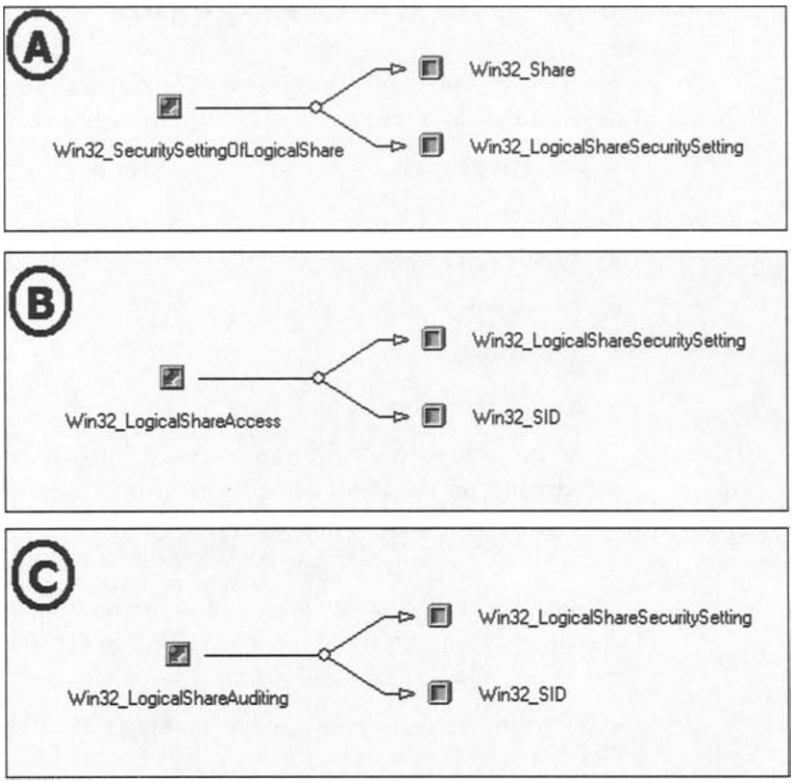


Figure 4.16 *Win32_LogicalShareSecuritySetting class associations.*

By using all these relationships, we decipher the security descriptor content by using the abstraction layer of the CIM repository. However, at no time do we look at the security descriptor directly. We always look at the security descriptor information through the relations defined in the CIM repository between different classes representing some items of its logical structure.

Another solution is to retrieve a security descriptor in a *Win32_SecurityDescriptor* instance. This solution is the best approach, because it provides a closer look at the security descriptor content. Moreover, this technique allows the implementation of a deciphering technique applicable to any security descriptor (i.e., file, Active Directory, registry key, etc.) and therefore applicable to any security descriptor access method.

4.6 Connecting to the manageable entities

Since the philosophy of this book is to be practical, the rest of this chapter is dedicated to the examination of a script managing security descriptors of:

- A file or a folder
- A share
- An Active Directory object
- An Exchange 2000 mailbox
- A registry key
- A CIM repository namespace

As we said, the technique to access and update the security descriptor will depend on the WMI capabilities. If it is impossible to get a security descriptor via WMI, and if ADSI represents a valid alternative to WMI, we will use ADSI to complement the WMI functionality.

Independent of the technique used to access and manage the security descriptor, the next script sample (called **WMIManageSD.Wsf**) implements the following operations:

- Perform the connection to the manageable entity (i.e., file, share, CIM repository namespace, etc.).
- Retrieve the security descriptor of that manageable entity (via WMI and/or ADSI).
- Convert the security descriptor to an ADSI structural representation if necessary.
- Decipher the security descriptor structural representation to view the security settings on the manageable entity.
- Update the security descriptor owner.
- Update the security descriptor group.
- Update the security descriptor control flags.
- Add Access Control Entries in the Access Control List.
- Remove Access Control Entries in the Access Control List.
- Reorder the Access Control Entries in the Access Control List.
- Update the security descriptor back to the manageable entity.

These various operations are implemented in Samples 4.2 through 4.61.

As we discover the script, we will see the techniques used to manage the various security descriptors. As with all scripts we have seen in previous chapters, this script will follow the same philosophy by exposing various command-line parameters to manage the security descriptors. The script exposes the following command-line parameters:

```
C:\>WMIManageSD.Wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Usage: WMIManageSD.Wsf [/FileSystem:value] [/Share:value] [/ADObject:value] [/E2KMailbox:value]
                       [/E2KStore[+|-]] [/RegistryKey:value] [/WMINameSpace:value]
                       [/ViewSD[+|-]]
                       [/Owner:value] [/Group:value] [/SDControls:value]
                       [/AddAce[+|-]] [/DelAce[+|-]]
                       [/Trustee:value] [/ACEMask:value] [/ACEType:value] [/ACEFlags:value]
                       [/ObjectType:value] [/InheritedObjectType:value]
                       [/SACL[+|-]] [/Decipher[+|-]] [/ADSI[+|-]] [/SIDResolutionDC[+|-]]
                       [/Machine:value] [/User:value] [/Password:value]

Options:

FileSystem          : Get the security descriptor of the specified file or directory path.
Share               : Get the security descriptor of the specified share name.
ADObject            : Get the security descriptor of the specified distinguished name AD object.
E2KMailbox          : Get the security descriptor of the Exchange 2000 mailbox
                      specified by AD user distinguished name.
E2KStore            : Specify if the security descriptor must come from the Exchange 2000 store.
RegistryKey         : Get the security descriptor of the specified registry key.
WMINameSpace        : Get the security descriptor of the specified WMI Name space.
ViewSD              : Decipher the security descriptor.
Owner               : Set the security descriptor owner.
Group               : Set the security descriptor group.
SDControls          : Set the security descriptor control flags.
AddAce              : Add a new ACE to the ACL.
DelAce              : Remove an existing ACE from the ACL.
Trustee             : Specify the ACE mask (granted user, group or machine account).
ACEMask             : Specify the ACE mask (granted rights).
ACEType             : Specify the ACE type (allow or deny the ACE mask).
ACEFlags            : Specify the ACE flags (ACE mask inheritance).
ObjectType          : Specify which object type, property set, or property an ACE refers to.
InheritedObjectType : Specify the GUID of an object that will inherit the ACE.
SACL                : Manage the System ACL (auditing) (default=Discretionary ACL).
Decipher            : Decipher the security descriptor.
ADSI                : Retrieve the security descriptor with ADSI.
SIDResolutionDC     : Domain Controller to use for SID resolution.
Machine             : Determine the WMI system to connect to. (default=LocalHost)
User                : Determine the UserID to perform the remote connection. (default=none)
Password            : Determine the password to perform the remote connection. (default=none)
```

The switch use is determined by the nature of the managed entity. During the script discovery, we will also examine the various command-line parameters to properly access, decipher, and update the respective security descriptors.

As usual, Sample 4.2 starts with the command-line parameters definition (skipped lines 13 through 153) and parsing (skipped lines 261 through 505). It is important to note that the script is initially written for Windows XP and Windows Server 2003. However, it is easily adaptable for Windows 2000 and Windows NT 4.0. The lines that must be used for Windows 2000 (and earlier versions) are commented out in the code. If you plan to use this script under Windows 2000, you must comment out the lines pointed to by a "Windows Server 2003 only" comment and remove the comment character of the lines pointed to by a "Windows 2000 only" comment. For instance, lines 188 through 192 and 199 through 203 illustrate the script adaptability to Windows 2000. We will see some more comments like this in the code, especially in the functions reading and updating the security descriptor. If you run under Windows NT 4.0, make sure that the latest update of WSH, WMI, and ADSI are installed on the managed systems. Once done, the script will use the same techniques under Windows NT 4.0 as under Windows 2000.

Sample 4.2 *The WMIManageSD.Wsf framework to manage security descriptors from the command line*

```
  1:<?xml version="1.0"?>
  .:
  8:<package>
  9:  <job>
 ..:
 13:    <runtime>
...:
153:    </runtime>
154:
155:    <script language="VBScript" src="..\Functions\SecurityInclude.vbs" />
156:    <script language="VBScript" src="..\Functions\GetSDFunction.vbs" />
157:    <script language="VBScript" src="..\Functions\SetSDFunction.vbs" />
158:    <script language="VBScript" src="..\Functions\DecipherWMISDFunction.vbs" />
159:    <script language="VBScript" src="..\Functions\DecipherADSISDFunction.vbs" />
160:
161:    <script language="VBScript" src="..\Functions\SetSDOwnerFunction.vbs" />
162:    <script language="VBScript" src="..\Functions\SetSDGroupFunction.vbs" />
163:
164:    <script language="VBScript" src="..\Functions\DecipherSDControlFlagsFunction.vbs" />
165:    <script language="VBScript" src="..\Functions\CalculateSDControlFlagsFunction.vbs" />
166:    <script language="VBScript" src="..\Functions\SetSDControlFlagsFunction.vbs" />
167:
168:    <script language="VBScript" src="..\Functions\DecipherACEFunction.vbs" />
169:    <script language="VBScript" src="..\Functions\CalculateACEFunction.vbs" />
170:
171:    <script language="VBScript" src="..\Functions\AddACEFunction.vbs" />
172:    <script language="VBScript" src="..\Functions\DelACEFunction.vbs" />
173:    <script language="VBScript" src="..\Functions\CreateDefaultSDFunction.vbs" />
174:
175:    <script language="VBScript" src="..\Functions\CreateTrusteeFunction.vbs" />
176:    <script language="VBScript" src="..\Functions\ReOrderACEFunction.vbs" />
```

```
177:    <script language="VBScript" src="..\Functions\ExtractUserIDFunction.vbs" />
178:    <script language="VBScript" src="..\Functions\ExtractUserDomainFunction.vbs" />
179:    <script language="VBScript" src="..\Functions\ConvertStringInArrayFunction.vbs" />
180:    <script language="VBScript" src="..\Functions\ConvertArrayInStringFunction.vbs" />
181:
182:    <script language="VBScript" src="..\Functions\DisplayFormattedSTDProperty Function.vbs" />
183:    <script language="VBScript" src="..\Functions\DisplayFormattedPropertyFunction.vbs" />
184:    <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
185:
186:    <reference object="ADs" version="1.0"/>
187:
188:    <!-- ***** Windows Server 2003 only ***** -->
189:    <object progid="ADsSecurityUtility" id="objADsSecurity"/>
190:
191:    <!-- ***** Windows 2000 only ***** -->
192:    <!-- <object progid="ADsSecurity" id="objADsSecurity"/> -->
193:
194:    <object progid="ADSIHelper.SDConversions" id="objADSIHelper"/>
195:
196:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
197:    <object progid="WbemScripting.SWbemNamedValueSet" id="objWMINamedValueSet" />
198:
199:    <!-- ***** Windows Server 2003 only ***** -->
200:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
201:
202:    <!-- ***** Windows 2000 only *****-->
203:    <!-- <object progid="SWbemDateTime.WSC" id="objWMIDateTime" /> -->
204:
205:    <script language="VBscript">
206:    <![CDATA[
...:
210:    ' ---------------------------------------------------------------------------
211:    Const cComputerName = "LocalHost"
212:    Const cWMICIMv2NameSpace = "root/cimv2"
213:    Const cWMIADNameSpace = "root\directory\LDAP"
...:
261:    ' ---------------------------------------------------------------------------
262:    ' Parse the command line parameters
263:    If WScript.Arguments.Named.Count = 0 Then
264:       WScript.Arguments.ShowUsage()
265:       WScript.Quit
266:    End If
267:
268:    strFileSystem = WScript.Arguments.Named("FileSystem")
269:    strShare = WScript.Arguments.Named("Share")
270:    strADsObjectDN = WScript.Arguments.Named("ADObject")
271:    strADsUserDN = WScript.Arguments.Named("E2KMailbox")
272:    strRegistryKey = WScript.Arguments.Named("RegistryKey")
273:    strWMINameSpace = WScript.Arguments.Named("WMINameSpace")
...:
495:    strUserID = WScript.Arguments.Named("User")
496:    If Len(strUserID) = 0 Then strUserID = ""
497:
498:    strPassword = WScript.Arguments.Named("Password")
499:    If Len(strPassword) = 0 Then strPassword = ""
500:
501:    strComputerName = WScript.Arguments.Named("Machine")
502:    If Len(strComputerName) = 0 Then strComputerName = cComputerName
503:
504:    strSIDResolutionDC = WScript.Arguments.Named("SIDResolutionDC")
```

```
 505:    If Len(strSIDResolutionDC) = 0 Then strSIDResolutionDC = strComputerName
 506:
 507:  Select Case intSDType
 508:' +-------------------------------------------------------------------------------------------+
 509:' | File or Folder                                                                            |
 510:' +-------------------------------------------------------------------------------------------+
 511:         Case cFileViaWMI
 512:' WMI technique retrieval ---------------------------------------------------------------------
 ...:
 584:
 585:         Case cFileViaADSI
 586:' ADSI technique retrieval --------------------------------------------------------------------
 ...:
 644:
 645:' +-------------------------------------------------------------------------------------------+
 646:' | Share                                                                                     |
 647:' +-------------------------------------------------------------------------------------------+
 648:         Case cShareViaWMI
 649:' WMI technique retrieval ---------------------------------------------------------------------
 ...:
 722:
 723:         Case cShareViaADSI
 724:' ADSI technique retrieval --------------------------------------------------------------------
 725:              ' Security descriptor access not implemented via ADSI under Windows 2000.
 726:
 727:              ' Windows Server 2003 only -----------------------------------------------------
 ...:
 787:
 788:' +-------------------------------------------------------------------------------------------+
 789:' | Active Directory object                                                                   |
 790:' +-------------------------------------------------------------------------------------------+
 791:         Case cActiveDirectoryViaWMI
 792:' WMI technique retrieval ---------------------------------------------------------------------
 ...:
 860:
 861:         Case cActiveDirectoryViaADSI
 862:' ADSI technique retrieval --------------------------------------------------------------------
 ...:
 933:
 934:' +-------------------------------------------------------------------------------------------+
 935:' | Exchange 2000 mailbox                                                                     |
 936:' +-------------------------------------------------------------------------------------------+
 937:         Case cExchange2000MailboxViaWMI
 938:' WMI technique retrieval ---------------------------------------------------------------------
....:
1006:
1007:         Case cExchange2000MailboxViaADSI
1008:' ADSI technique retrieval --------------------------------------------------------------------
....:
1077:
1078:         Case cExchange2000MailboxViaCDOEXM
1079:' CDOEXM technique retrieval ------------------------------------------------------------------
....:
1147:
1148:' +-------------------------------------------------------------------------------------------+
1149:' | Registry key                                                                              |
1150:' +-------------------------------------------------------------------------------------------+
1151:         Case cRegistryViaWMI
1152:' WMI technique retrieval ---------------------------------------------------------------------
1153:              ' Security descriptor access not implemented via WMI.
```

```
1154:
1155:         Case cRegistryViaADSI
1156:' ADSI technique retrieval ------------------------------------------------------------------
....:
1213:
1214:' +-------------------------------------------------------------------------------------------+
1215:' | CIM repository namespace                                                                  |
1216:' +-------------------------------------------------------------------------------------------+
1217:         Case cWMINameSpaceViaWMI
1218:' WMI technique retrieval --------------------------------------------------------------------
....:
1288:
1289:         Case cWMINameSpaceViaADSI
1290:' ADSI technique retrieval -------------------------------------------------------------------
1291:               ' Security descriptor access not implemented via ADSI.
1292:    End Select
1293:
1294:    ]]>
1295:    </script>
1296:  </job>
1297:</package>
```

Sample 4.2 is the framework used by **WMIManageSD.Wsf** to manage security descriptors. We will examine all functions included (lines 155 through 184) and the code contained in this framework (lines 507 through 1292) in the following sections. Sample 4.2 is organized in a "Select Case" structure. Each "Case" corresponds to the management of a security descriptor with a specific access method. We have a "Case" section for:

- Security descriptor from files and folders accessed by WMI (lines 511 through 584)
- Security descriptor from files and folders accessed by ADSI (lines 585 through 644)
- Security descriptor from shares accessed by WMI (lines 648 through 722)
- Security descriptor from shares accessed by ADSI (lines 723 to 787)
- Security descriptor from Active Directory objects accessed by WMI (lines 791 through 860)
- Security descriptor from Active Directory objects accessed by ADSI (lines 861 through 933)
- Security descriptor from Exchange 2000 mailboxes accessed by WMI (lines 937 through 1006)
- Security descriptor from Exchange 2000 mailboxes accessed by ADSI (lines 1007 through 1077)

- Security descriptor from Exchange 2000 mailboxes accessed by CDOEXM (lines 1078 through 1147)
- Security descriptor from registry keys accessed by WMI (lines 1151 through 1154)
- Security descriptor from registry keys accessed by ADSI (lines 1155 through 1213)
- Security descriptor from CIM repository namespaces accessed by WMI (lines 1217 through 1288)
- Security descriptor from CIM repository namespaces accessed by ADSI (lines 1289 through 1292)

Before trying to change the content of a security descriptor, let's first see how we can retrieve a security descriptor from a manageable instance and how we can decipher it. Since WMI represents real-world objects with some specific classes, we must examine the technique used to retrieve the security descriptor, based on the nature and the class capabilities representing the real-world object. Let's start with the file system objects first.

4.6.1 Connecting to file and folder security descriptors

4.6.1.1 Connecting to files and folders with WMI

Accessing file and folder security descriptors with WMI involves establishing the WMI connection to the **Root\CIMv2** namespace (Sample 4.3), since it contains the *Win32_LogicalFileSecuritySetting* class, which exposes a method to retrieve the security descriptor. This connection is completed from line 513 through 520. At line 515, the addition of the **SeSecurityPrivilege** privilege to the **SWBemLocator** object is required to access the SACL part of the security descriptor.

To execute the Sample 4.3 code portion, the following command lines must be used:

```
C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory
C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory\MyFile.Txt
```

Sample 4.3 *Connecting to files and folders with WMI (Part I)*

```
...:
...:
...:
508:' +-------------------------------------------------------------------------------+
509:' | File or Folder                                                                |
510:' +-------------------------------------------------------------------------------+
```

```
511:         Case cFileViaWMI
512:' WMI technique retrieval ------------------------------------------------------------------
513:             objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
514:             objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
515:             objWMILocator.Security_.Privileges.AddAsString "SeSecurityPrivilege", True
516:
517:             Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
518:                                                              cWMICIMv2NameSpace, _
519:                                                              strUserID, _
520:                                                              strPassword)
...:
523:             Set objSD = GetSecurityDescriptor (objWMIServices, _
524:                                                strFileSystem, _
525:                                                intSDType)
526:             If boolAddAce Then
527:                Set objSD = AddACE (objWMIServices, strSIDResolutionDC, _
528:                                    strUserID, strPassword, _
529:                                 objSD, _
530:                                 strTrustee, _
531:                                 intACEType, _
532:                                 intACEMask, _
533:                                 intACEFlags, _
534:                                 intACLtype, _
535:                                 vbNull, _
536:                                 vbNull, _
537:                                 intSDType)
538:             End If
539:
540:             If boolDelAce Then
541:                Set objSD = DelACE (objWMIServices, strSIDResolutionDC, _
542:                                    strUserID, strPassword, _
543:                                 objSD, _
544:                                 strTrustee, _
545:                                 intACLtype, _
546:                                 intSDType)
547:             End If
548:
549:             If boolOwner Then
550:                Set objSD = SetSDOwner(strSIDResolutionDC, strUserID, strPassword, _
551:                                    objSD, strOwner, intSDType)
552:             End If
553:
554:             If boolGroup Then
555:                Set objSD = SetSDGroup(strSIDResolutionDC, strUserID, strPassword, _
556:                                    objSD, strGroup, intSDType)
557:             End If
558:
559:             If boolSDControlFlags Then
560:                Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
561:             End If
562:
563:             If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
564:                If boolAddAce Or boolDelAce Then
565:                   Set objSD = ReOrderACE(objWMIServices, objSD, intSDType)
566:                End If
567:
568:                SetSecurityDescriptor objWMIServices, _
569:                                    objSD, _
570:                                    strFileSystem, _
571:                                    intSDType
```

```
572:            End If
573:
574:            If boolViewSD Then
575:               WScript.Echo
576:               DecipherWMISecurityDescriptor objSD, _
577:                                             intSDType, _
578:                                             "", _
579:                                             boolDecipher
580:            End If
...:
585:        Case cFileViaADSI
...:
...:
...:
```

Once the WMI connection completes, the script invokes a series of sub-functions:

- Lines 523 through 525 read the security descriptor from the selected file or folder with the GetSecurityDescriptor() function.
- Lines 526 through 538 add an ACE in the security descriptor with the AddAce() function, if specified to do so on the command line.
- Lines 540 through 547 remove an ACE to the security descriptor with the DelAce() function, if specified to do so on the command line.
- Lines 549 through 552 set the owner from the security descriptor with the SetSDOwner() function, if specified to do so on the command line.
- Lines 554 through 557 set the group in the security descriptor with the SetSDGroup(), if specified to do so on the command line.
- Lines 559 through 561 set the security descriptor control flags with the SetSDControlFlags() function, if specified to do so on the command line.
- Lines 564 through 566 reorder the security descriptor ACE in the discretionary ACL with the ReOrderAce() function, if an ACE addition or removal is performed.
- Lines 568 through 571 update the security descriptor on the selected file or folder with the SetSecurityDescriptor() function.
- Lines 576 through 579 decipher and display the security descriptor settings with the DecipherWMISecurityDescriptor() function, if specified to do so on the command line. Note that this function can be replaced by the DecipherADSISecurityDescriptor() function, based on the object model used to represent the security descriptor. In

this particular example, the file or folder security descriptor accessed with the WMI access method is represented in the WMI object model. So, the security descriptor deciphering will be completed with the DecipherWMISecurityDescriptor() function.

Note that the code structure is always the same for any security descriptor type (i.e., files, Active Directory objects, CIM repository namespace) and access method technique (WMI or ADSI). As we will see further, only the WMI or the ADSI connection logic, with some parameters passed to the subfunctions, will vary. The subfunctions (i.e., AddAce(), DelAce(), ReOrderAce(), etc.) will manage the security descriptor specifics according to their nature and format. For instance, in Sample 4.3, the WMI security descriptor access method retrieves the security descriptor in a WMI representation, which implies the use of a *Win32_SecurityDescriptor* instance. In order to properly manipulate this security descriptor representation in the subfunctions, it is necessary to pass the WMI connection made to the **Root\CIMv2** namespace to the subfunctions, since this namespace contains the *Win32_SecurityDescriptor*, *Win32_ACE*, and *Win32_Trustee* class definitions.

4.6.1.2 Connecting to files and folders with ADSI

When the security descriptor is represented in the ADSI object model, it is not necessary to pass an object representing the connection to the entity owning the security descriptor. Sample 4.4 illustrates this approach. We retrieve the exact same structure as Sample 4.3 ("Connecting to files and folders with WMI [Part I]"), while the access method to the file or the folder security descriptor is made with ADSI. In this case, no specific connection is made to the file or folder, because it uses a different tactic to get access to the entity owning the security descriptor. Based on the operating system, the script will use:

- The **ADsSecurityUtility** object instantiated at line 189 in Sample 4.2 ("The **WMIManageSD.Wsf** framework to manage security descriptors from the command line"), if you run under Windows XP or Windows Server 2003. This object is only available under Windows XP and Windows Server 2003 and is implemented by the ADSI **IADsSecurityUtility** interface.
- The **ADsSecurity** object instantiated at line 192 in Sample 4.2, if you run under Windows NT or Windows 2000. Note that the **ADsSecurity** object requires the registration of the **ADsSecurity.DLL**, which is available from the Windows 2000 Resource Kit (or the ADSI SDK

for Windows NT 4.0). It is very important to know that the **ADsSecurity.DLL** is not designed to retrieve the SACL part of a security descriptor. This is a limitation of the **ADsSecurity.DLL**.

As shown in Sample 4.4, the first parameter of the GetSecurityDescriptor() function is set to Null (line 587). The ADSI security descriptor access method for a file or a folder via ADSI does not need an object representing the connection to the examined file or folder, because the **ADsSecurityUtility** or **ADsSecurity** object uses an encapsulated logic running in the user security context. Because the ADSI access technique retrieves the security descriptor in an ADSI representation and not in a WMI presentation, there is no need to access the **Root\CIMv2** namespace. Therefore, any subsequent subfunction calls (i.e., AddAce(), DelAce(), ReOrderAce(), etc.) use a Null parameter for the WMI connection settings. Next, the security descriptor deciphering will be handled by DecipherADSISecurityDescriptor() function (lines 638 through 640).

The following command lines will invoke the Sample 4.4 code portion:

```
C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /ADSI+

C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory\MyFile.Txt /ADSI+
```

Sample 4.4 *Connecting to files and folders with ADSI (Part II)*

```
...:
...:
...:
585:         Case cFileViaADSI
586:' ADSI technique retrieval ------------------------------------------------------------------------
587:              Set objSD = GetSecurityDescriptor (vbNull, _
588:                                                 strFileSystem, _
589:                                                 intSDType)
590:              If boolAddAce Then
591:                 Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
592:                                     objSD, _
593:                                     strTrustee, _
594:                                     intACEType, _
595:                                     intACEMask, _
596:                                     intACEFlags, _
597:                                     intACLtype, _
598:                                     vbNull, _
599:                                     vbNull, _
600:                                     intSDType)
601:              End If
602:
603:              If boolDelAce Then
604:                 Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
605:                                     objSD, _
606:                                     strTrustee, _
607:                                     intACLtype, _
608:                                     intSDType)
609:              End If
```

```
610:
611:            If boolOwner Then
612:               Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
613:                                      objSD, strOwner, intSDType)
614:            End If
615:
616:            If boolGroup Then
617:               Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
618:                                      objSD, strGroup, intSDType)
619:            End If
620:
621:            If boolSDControlFlags Then
622:               Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
623:            End If
624:
625:            If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
626:               If boolAddAce Or boolDelAce Then
627:                  Set objSD = ReOrderACE(vbNull, objSD, intSDType)
628:               End If
629:
630:               SetSecurityDescriptor vbNull, _
631:                                     objSD, _
632:                                     strFileSystem, _
633:                                     intSDType
634:            End If
635:
636:            If boolViewSD Then
637:               WScript.Echo
638:               DecipherADSISecurityDescriptor objSD, _
639:                                              intSDType, _
640:                                              boolDecipher
641:            End If
...:
644:
...:
...:
...:
```

4.6.2 Connecting to file system share security descriptors

4.6.2.1 *Connecting to file system shares with WMI*

As with files and folders, the WMI logic to access a share security descriptor is exactly the same (see Sample 4.3, "Connecting to files and folders with WMI [Part I]"). Once the WMI connection to the **Root\CIMv2** namespace is established (lines 650 through 656 in Sample 4.5), the parameters passed to the subfunctions managing the security descriptor are the same. This similarity comes from two factors:

- The security descriptor access method is based on WMI.
- The security descriptor is represented by a *Win32_SecurityDescriptor* instance, which implies the use of the DecipherWMISecurity-

Descriptor() function to decipher the security descriptor (lines 714 through 717).

Of course, as we manage a share security descriptor, we must note some small differences as well:

- The GetSecurityDescriptor() takes a share name instead of a file or folder path (line 663).
- The modification of the security descriptor owner, group, or control flags is not applicable to a share security descriptor (lines 685 through 699).
- The SACL is not supported on a security descriptor share.

Once these differences are taken into consideration (commented out lines 685 through 699), the code logic for a share is exactly the same as before. The following command line will invoke the Sample 4.5 code portion:

```
C:\>WMIManageSD.Wsf /Share:MyDirectory
```

Sample 4.5 *Connecting to shares with WMI (Part III)*

```
...:
...:
...:
644:
645:' +-------------------------------------------------------------------------------------+
646:' | Share                                                                               |
647:' +-------------------------------------------------------------------------------------+
648:         Case cShareViaWMI
649:' WMI technique retrieval ------------------------------------------------------------
650:              objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
651:              objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
652:
653:              Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
654:                                                               cWMICIMv2NameSpace, _
655:                                                               strUserID, _
656:                                                               strPassword)
...:
659:              Set objSD = GetSecurityDescriptor (objWMIServices, _
660:                                                 strShare, _
661:                                                 intSDType)
662:              If boolAddAce Then
663:                 Set objSD = AddACE (objWMIServices, strSIDResolutionDC, _
664:                                     strUserID, strPassword, _
665:                                 objSD, _
666:                                 strTrustee, _
667:                                 intACEType, _
668:                                 intACEMask, _
669:                                 intACEFlags, _
670:                                 intACLtype, _
671:                                 vbNull, _
672:                                 vbNull, _
```

```
673:                                    intSDType)
674:            End If
675:
676:            If boolDelAce Then
677:               Set objSD = DelACE (objWMIServices, strSIDResolutionDC, _
678:                                      strUserID, strPassword, _
679:                                   objSD, _
680:                                   strTrustee, _
681:                                   intACLtype, _
682:                                   intSDType)
683:            End If
684:
685:            ' Not supported for a share security descriptor.
686:            '
687:            ' If boolOwner Then
688:            '    Set objSD = SetSDOwner(strSIDResolutionDC, strUserID, strPassword, _
689:            '                           objSD, strOwner, intSDType)
690:            ' End If
691:            '
692:            ' If boolGroup Then
693:            '    Set objSD = SetSDGroup(strSIDResolutionDC, strUserID, strPassword, _
694:            '                           objSD, strGroup, intSDType)
695:            ' End If
696:            '
697:            ' If boolSDControlFlags Then
698:            '    Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
699:            ' End If
700:
701:            If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
702:               If boolAddAce Or boolDelAce Then
703:                  Set objSD = ReOrderACE(objWMIServices, objSD, intSDType)
704:               End If
705:
706:               SetSecurityDescriptor objWMIServices, _
707:                                     objSD, _
708:                                     strShare, _
709:                                     intSDType
710:            End If
711:
712:            If boolViewSD Then
713:               WScript.Echo
714:               DecipherWMISecurityDescriptor objSD, _
715:                                             intSDType, _
716:                                             "", _
717:                                             boolDecipher
718:            End If
...:
723:        Case cShareViaADSI
...:
...:
...:
```

4.6.2.2 *Connecting to file system shares with ADSI*

Connecting to a share security descriptor with ADSI is only possible under Windows XP or Windows Server 2003, because the ADSI **ADsSecurityUtility** object is only available for these platforms (see Sample 4.6). The **ADsSecurity.DLL** is not designed for this purpose. Since we use the ADSI

ADsSecurityUtility object to retrieve the security descriptor, it is not necessary to establish a connection to the manageable entity. Basically, it follows the same logic as Sample 4.4 ("Connecting to files and folders with ADSI [Part II]") but with the same restrictions as the WMI techniques: Owner, group, control flags, and SACL updates are not applicable to a share. The retrieved security descriptor is represented in the ADSI object model, which implies the use of the DecipherADSISecurityDescriptor() function (lines 781 through 783). If you run under Windows Server 2003 or Windows XP, you can use the following command line to execute this portion of the script:

```
C:\>WMIManageSD.Wsf /Share:MyDirectory /ADSI+
```

Sample 4.6 *Connecting to shares with ADSI (Part IV)*

```
...:
...:
...:
723:         Case cShareViaADSI
724:' ADSI technique retrieval ------------------------------------------------------------------
725:              ' Security descriptor access not implemented via ADSI under Windows 2000.
726:
727:              ' Windows Server 2003 only ----------------------------------------------------
728:              Set objSD = GetSecurityDescriptor (vbNull, _
729:                                                 strShare, _
730:                                                 intSDType)
731:              If boolAddAce Then
732:                 Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
733:                                     objSD, _
734:                                     strTrustee, _
735:                                     intACEType, _
736:                                     intACEMask, _
737:                                     intACEFlags, _
738:                                     intACLtype, _
739:                                     vbNull, _
740:                                     vbNull, _
741:                                     intSDType)
742:              End If
743:
744:              If boolDelAce Then
745:                 Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
746:                                     objSD, _
747:                                     strTrustee, _
748:                                     intACLtype, _
749:                                     intSDType)
750:              End If
751:
752:              ' Not supported for a share security descriptor.
753:              '
754:              ' If boolOwner Then
755:              '    Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
756:              '                           objSD, strOwner, intSDType)
757:              ' End If
758:              '
```

```
759:            ' If boolGroup Then
760:            '    Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
761:            '                           objSD, strGroup, intSDType)
762:            ' End If
763:            '
764:            ' If boolSDControlFlags Then
765:            '    Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
766:            ' End If
767:
768:            If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
769:               If boolAddAce Or boolDelAce Then
770:                  Set objSD = ReOrderACE(vbNull, objSD, intSDType)
771:               End If
772:
773:               SetSecurityDescriptor vbNull, _
774:                                     objSD, _
775:                                     strShare, _
776:                                     intSDType
777:            End If
778:
779:            If boolViewSD Then
780:               WScript.Echo
781:               DecipherADSISecurityDescriptor objSD, _
782:                                              intSDType, _
783:                                              boolDecipher
784:            End If
...:
787:
...:
...:
...:
```

4.6.3 Connecting to Active Directory object security descriptors

4.6.3.1 *Connecting to Active Directory objects with WMI*

Connecting to an Active Directory object with WMI follows the same rules as Sample 4.3 ("Connecting to files and folders with WMI [Part I]"). However, we must connect to the **Root\Directory\LDAP** CIM repository namespace, instead of the **Root\CIMv2** namespace (lines 769 through 799), in order to get access to the WMI classes representing Active Directory object classes. As mentioned previously in section 4.4 ("Which access technique to use? Which security descriptor representation do we obtain?"), the WMI security descriptor access method of an Active Directory object retrieves the security descriptor in a binary format. The conversion of the security descriptor to an ADSI security descriptor representation is made in the GetSecurityDescriptor() function. This is why, even if the security descriptor access method is based on WMI, the subsequent subfunctions (i.e., AddAce() at line 806, DelAce() at line 819, and ReOrderAce() at line 842, to name a few) do not use the WMI connection settings. Instead, these

functions manipulate the security descriptor in its ADSI representation. This implies that the security descriptor deciphering technique will be handled by the DecipherADSISecurityDescriptor() function (lines 853 through 855).

Another point to note is the format of the Active Directory distinguished name. Because the script can retrieve any object type from Active Directory, and because WMI requires the class of the object, it is mandatory to specify the object class on the command line. This must be done as follows:

```
C:\>WMIManageSD.Wsf /ADObject:"user;CN=MyUser,CN=Users,DC=LissWare,DC=Net"
```

The script will properly decode the class and the **distinguishedName** in the GetSecurityDescriptor() function.

Sample 4.7 *Connecting to Active Directory objects with WMI (Part V)*

```
...:
...:
...:
787:
788:' +-------------------------------------------------------------------------------------+
789:' | Active Directory object                                                             |
790:' +-------------------------------------------------------------------------------------+
791:         Case cActiveDirectoryViaWMI
792:' WMI technique retrieval ------------------------------------------------------------
793:              objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
794:              objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
795:
796:              Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
797:                                                               cWMIADNameSpace, _
798:                                                               strUserID, _
799:                                                               strPassword)
...:
802:              Set objSD = GetSecurityDescriptor (objWMIServices, _
803:                                                 strADsObjectDN, _
804:                                                 intSDType)
805:              If boolAddAce Then
806:                 Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
807:                                     objSD, _
808:                                     strTrustee, _
809:                                     intACEType, _
810:                                     intACEMask, _
811:                                     intACEFlags, _
812:                                     intACLtype, _
813:                                     strObjectType, _
814:                                     strInheritedObjectType, _
815:                                     intSDType)
816:              End If
817:
818:              If boolDelAce Then
819:                 Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
820:                                     objSD, _
821:                                     strTrustee, _
```

```
822:                                    intACLtype, _
823:                                    intSDType)
824:            End If
825:
826:            If boolOwner Then
827:               Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
828:                                      objSD, strOwner, intSDType)
829:            End If
830:
831:            If boolGroup Then
832:               Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
833:                                      objSD, strGroup, intSDType)
834:            End If
835:
836:            If boolSDControlFlags Then
837:               Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
838:            End If
839:
840:            If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
841:               If boolAddAce Or boolDelAce Then
842:                  Set objSD = ReOrderACE(vbNull, objSD, intSDType)
843:               End If
844:
845:               SetSecurityDescriptor objWMIServices, _
846:                                     objSD, _
847:                                     strADsObjectDN, _
848:                                     intSDType
849:            End If
850:
851:            If boolViewSD Then
852:               WScript.Echo
853:               DecipherADSISecurityDescriptor objSD, _
854:                                              intSDType, _
855:                                              boolDecipher
856:            End If
...:
861:        Case cActiveDirectoryViaADSI
...:
...:
...:
```

4.6.3.2 *Connecting to Active Directory objects with ADSI*

To retrieve the security descriptor of an Active Directory object with ADSI (see Sample 4.8), it is necessary to connect to the desired Active Directory object first. This can be done in the current user security context (lines 871 and 872) or with different credentials (lines 864 through 869). The connection security context is determined by the presence of the **/UserID** switch on the command line, which determines the content of the **strUserID** variable. Under Windows 2000, we do not use the **ADsSecurity** object method, even if it can retrieve the security descriptor from an Active Directory object. Coding the logic directly with the ADSI base objects, instead of using another COM object encapsulating its own logic, gives us more control over the coding. Moreover, this technique is applicable to any platform

(Windows 2000, Windows Server 2003, Windows XP). This is why we must perform the connection to the Active Directory object. In the case of a file, a folder, or a share, this is done by the logic encapsulated in the **ADsSecurityUtility** or **ADsSecurity** objects.

Once the connection is established, the script invokes the GetSecurityDescriptor() function with the object referring to the Active Directory object connection (line 875). The **distinguishedName** of the object is also passed to the function (line 876). The following command line will execute this portion of the script:

```
C:\>WMIManageSD.Wsf /ADObject:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /ADSI+
```

Note that with the ADSI access method, we do not provide the object class with the **distinguishedName**, as done with the WMI access method. ADSI is able to determine the object class by itself. Once the security descriptor is retrieved (line 875), since it is represented in the ADSI object model, the subsequent functions do not need the connection parameters, unlike the case with WMI. The security descriptor is deciphered with the DecipherADSISecurityDescriptor() function (lines 926 through 928).

Sample 4.8 *Connecting to Active Directory objects with ADSI (Part VI)*

```
...:
...:
...:
861:         Case cActiveDirectoryViaADSI
862:' ADSI technique retrieval ------------------------------------------------------------------------
863:               If Len (strUserID) Then
864:                  Set objNS = GetObject("LDAP:")
865:                  Set objADsObject = objNS.OpenDSObject("LDAP://" & strComputerName & _
866:                                                        "/" & strADsObjectDN, _
867:                                                        strUserID, _
868:                                                        strPassword, _
869:                                                        ADS_SECURE_AUTHENTICATION)
870:               Else
871:                  Set objADsObject = GetObject("LDAP://" & strComputerName & "/" & _
872:                                               strADsObjectDN)
873:               End If
874:
875:               Set objSD = GetSecurityDescriptor (objADsObject, _
876:                                                  strADsObjectDN, _
877:                                                  intSDType)
878:               If boolAddAce Then
879:                  Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
880:                                      objSD, _
881:                                      strTrustee, _
882:                                      intACEType, _
883:                                      intACEMask, _
884:                                      intACEFlags, _
885:                                      intACLtype, _
886:                                      strObjectType, _
887:                                      strInheritedObjectType, _
```

```
888:                                intSDType)
889:          End If
890:
891:          If boolDelAce Then
892:             Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
893:                                 objSD, _
894:                                 strTrustee, _
895:                                 intACLtype, _
896:                                 intSDType)
897:          End If
898:
899:          If boolOwner Then
900:             Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
901:                                    objSD, strOwner, intSDType)
902:          End If
903:
904:          If boolGroup Then
905:             Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
906:                                    objSD, strGroup, intSDType)
907:          End If
908:
909:          If boolSDControlFlags Then
910:             Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
911:          End If
912:
913:          If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
914:             If boolAddAce Or boolDelAce Then
915:                Set objSD = ReOrderACE(vbNull, objSD, intSDType)
916:             End If
917:
918:             SetSecurityDescriptor objADsObject, _
919:                                   objSD, _
920:                                   strADsObjectDN, _
921:                                   intSDType
922:          End If
923:
924:          If boolViewSD Then
925:             WScript.Echo
926:             DecipherADSISecurityDescriptor objSD, _
927:                                            intSDType, _
928:                                            boolDecipher
929:          End If
...:
933:
...:
...:
...:
```

4.6.4 Connecting to Exchange 2000 mailbox security descriptors

To get access to the security descriptor set on an Exchange 2000 mailbox, it is possible to use three techniques: WMI, ADSI, or CDOEXM. Although all techniques can retrieve and view the security descriptor settings, only the CDOEXM technique can perform an update of the Exchange 2000 security descriptor correctly, because the CDOEXM technique is the only one

updating the security descriptor located in the Exchange 2000 store. We will deal with this aspect when updating the Exchange 2000 security descriptor in section 4.14.4.3 ("Updating Exchange 2000 mailbox security descriptors with CDOEXM").

4.6.4.1 Connecting to Exchange 2000 mailbox security descriptor with WMI

Sample 4.9 illustrates the WMI technique. Basically, this is exactly the same as the technique used to retrieve the security descriptor from an Active Directory object, simply because the Exchange 2000 mailbox security descriptor is available from Active Directory. The only difference resides in the object class that is accessed, since only **user** objects can have an Exchange 2000 mailbox. That's the reason why the **distinguishedName** given on the command line does not require the object class anymore. For instance, the following command line will retrieve the Exchange 2000 mailbox security descriptor with WMI:

```
C:\>WMIManageSD.Wsf /E2KMailbox:"CN=MyUser,CN=Users,DC=LissWare,DC=Net"
```

However, the GetSecurityDescriptor() function will look for a specific attribute containing the Exchange 2000 mailbox security descriptor. We will see this in detail when examining the GetSecurityDescriptor() function as it accesses an Exchange 2000 mailbox security descriptor in section 4.7.4. Except for this difference encapsulated in the GetSecurityDescriptor() function, Sample 4.9 is the same as Sample 4.7 ("Connecting to Active Directory objects with WMI [Part V]").

Sample 4.9 *Connecting to Exchange 2000 mailbox information with WMI (Part VII)*

```
...:
...:
...:
933:
934:' +-------------------------------------------------------------------------------------+
935:' | Exchange 2000 mailbox                                                               |
936:' +-------------------------------------------------------------------------------------+
937:         Case cExchange2000MailboxViaWMI
938:' WMI technique retrieval ------------------------------------------------------------
939:              objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
940:              objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
941:
942:              Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
943:                                                               cWMIADNameSpace, _
944:                                                               strUserID, _
945:                                                               strPassword)
...:
948:              Set objSD = GetSecurityDescriptor (objWMIServices, _
949:                                                 strADsUserDN, _
950:                                                 intSDType)
```

```
            If boolAddAce Then
               Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
                                   objSD, _
                                   strTrustee, _
                                   intACEType, _
                                   intACEMask, _
                                   intACEFlags, _
                                   intACLtype, _
                                   vbNull, _
                                   vbNull, _
                                   intSDType)
            End If

            If boolDelAce Then
               Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
                                   objSD, _
                                   strTrustee, _
                                   intACLtype, _
                                   intSDType)
            End If

            If boolOwner Then
               Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
                                      objSD, strOwner, intSDType)
            End If

            If boolGroup Then
               Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
                                      objSD, strGroup, intSDType)
            End If

            If boolSDControlFlags Then
               Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
            End If

            If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
               If boolAddAce Or boolDelAce Then
                  Set objSD = ReOrderACE(vbNull, objSD, intSDType)
               End If

               SetSecurityDescriptor objWMIServices, _
                                     objSD, _
                                     strADsUserDN, _
                                     intSDType
            End If

            If boolViewSD Then
               WScript.Echo
               DecipherADSISecurityDescriptor objSD, _
                                              intSDType, _
                                              boolDecipher
            End If
....:
        Case cExchange2000MailboxViaADSI
....:
....:
....:
```

4.6.4.2 Connecting to Exchange 2000 mailbox security descriptor with ADSI

As with Sample 4.8 ("Connecting to Active Directory objects with ADSI [Part VI]"), Sample 4.10 follows the exact same logic. The security descriptor specificities are treated in the subfunctions. The following command line executes the Sample 4.10 code portion:

```
C:\>WMIManageSD.Wsf /E2KMailbox:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /ADSI+
```

Sample 4.10 *Connecting to Exchange 2000 mailbox information with ADSI (Part VIII)*

```
....:
....:
....:
1007:         Case cExchange2000MailboxViaADSI
1008:' ADSI technique retrieval ------------------------------------------------------------------
1009:              If Len (strUserID) Then
1010:                 Set objNS = GetObject("LDAP:")
1011:                 Set objADsObject = objNS.OpenDSObject("LDAP://" & strComputerName, _
1012:                                                       strUserID, _
1013:                                                       strPassword, _
1014:                                                       ADS_SECURE_AUTHENTICATION)
1015:              Else
1016:                 Set objADsObject = GetObject("LDAP://" & strComputerName & "/" & strADsUserDN)
1017:              End If
1018:
1019:              Set objSD = GetSecurityDescriptor (objADsObject, _
1020:                                                 strADsUserDN, _
1021:                                                 intSDType)
1022:              If boolAddAce Then
1023:                 Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
1024:                                     objSD, _
1025:                                     strTrustee, _
1026:                                     intACEType, _
1027:                                     intACEMask, _
1028:                                     intACEFlags, _
1029:                                     intACLtype, _
1030:                                     vbNull, _
1031:                                     vbNull, _
1032:                                     intSDType)
1033:              End If
1034:
1035:              If boolDelAce Then
1036:                 Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
1037:                                     objSD, _
1038:                                     strTrustee, _
1039:                                     intACLtype, _
1040:                                     intSDType)
1041:              End If
1042:
1043:              If boolOwner Then
1044:                 Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
1045:                                        objSD, strOwner, intSDType)
1046:              End If
1047:
1048:              If boolGroup Then
1049:                 Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
```

```
1050:                                    objSD, strGroup, intSDType)
1051:          End If
1052:
1053:          If boolSDControlFlags Then
1054:             Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
1055:          End If
1056:
1057:          If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
1058:             If boolAddAce Or boolDelAce Then
1059:                Set objSD = ReOrderACE(vbNull, objSD, intSDType)
1060:             End If
1061:
1062:             SetSecurityDescriptor objADsObject, _
1063:                                   objSD, _
1064:                                   strADsUserDN, _
1065:                                   intSDType
1066:          End If
1067:
1068:          If boolViewSD Then
1069:             WScript.Echo
1070:             DecipherADSISecurityDescriptor objSD, _
1071:                                            intSDType, _
1072:                                            boolDecipher
1073:          End If
....:
1078:    Case cExchange2000MailboxViaCDOEXM
....:
....:
....:
```

4.6.4.3 *Connecting to Exchange 2000 mailbox security descriptor with CDOEXM*

CDOEXM provides objects and interfaces for the management of many Exchange 2000 components. For instance, with CDOEXM you can configure Exchange Servers and stores, mount and dismount stores, and create and configure mailboxes. CDOEXM is more than an extension for ADSI; it is also an extension for Collaboration Data Object for Exchange 2000 (CDOEX). At the server level, CDOEXM retrieves specific information about the server itself, as well as the Storage Groups present on the server and stores created in the Storage Groups. From an Active Directory user object point of view, CDOEXM exposes properties and methods to manage the Exchange 2000 mailbox. CDOEXM is an important companion to ADSI and CDOEX when working with Exchange 2000. Our unique interest for the CDOEXM technique for the script purpose resides in its capability to update the security descriptor in the Exchange store (see section 4.13.4.3, "Updating Exchange 2000 mailbox security descriptors with CDOEXM").

Sample 4.11 refers to the CDOEXM security descriptor access technique. Basically, this is the same technique as Sample 4.10, since CDOEXM acts as an extension for ADSI. Again, the security descriptor

specificities are managed in the GetSecurityDescriptor() function. The following command line executes the Sample 4.11 code portion:

```
C:\>WMIManageSD.Wsf /E2KMailbox:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /E2KStore+
```

Sample 4.11 *Connecting to Exchange 2000 mailbox information with CDOEXM (Part IX)*

```
...:
...:
...:
.078:         Case cExchange2000MailboxViaCDOEXM
.079:' CDOEXM technique retrieval ------------------------------------------------------------------
.080:               If Len (strUserID) Then
.081:                  Set objNS = GetObject("LDAP:")
.082:                  Set objADsObject = objNS.OpenDSObject("LDAP://" & strADsUserDN, _
.083:                                                        strUserID, _
.084:                                                        strPassword, _
.085:                                                        ADS_SECURE_AUTHENTICATION)
.086:               Else
.087:                  Set objADsObject = GetObject("LDAP://" & strADsUserDN)
.088:               End If
.089:               Set objSD = GetSecurityDescriptor (objADsObject, _
.090:                                                  strADsUserDN, _
.091:                                                  intSDType)
.092:               If boolAddAce Then
.093:                  Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
.094:                                      objSD, _
.095:                                      strTrustee, _
.096:                                      intACEType, _
.097:                                      intACEMask, _
.098:                                      intACEFlags, _
.099:                                      intACLtype, _
.100:                                      vbNull, _
.101:                                      vbNull, _
.102:                                      intSDType)
.103:               End If
.104:
.105:               If boolDelAce Then
.106:                  Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
.107:                                      objSD, _
.108:                                      strTrustee, _
.109:                                      intACLtype, _
.110:                                      intSDType)
.111:               End If
.112:
.113:               If boolOwner Then
.114:                  Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
.115:                                         objSD, strOwner, intSDType)
.116:               End If
.117:
.118:               If boolGroup Then
.119:                  Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
.120:                                         objSD, strGroup, intSDType)
.121:               End If
.122:
.123:               If boolSDControlFlags Then
.124:                  Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
.125:               End If
```

```
1126:
1127:          If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
1128:             If boolAddAce Or boolDelAce Then
1129:                Set objSD = ReOrderACE(vbNull, objSD, intSDType)
1130:             End If
1131:
1132:             SetSecurityDescriptor objADsObject, _
1133:                                   objSD, _
1134:                                   strADsUserDN, _
1135:                                   intSDType
1136:          End If
1137:
1138:          If boolViewSD Then
1139:             WScript.Echo
1140:             DecipherADSISecurityDescriptor objSD, _
1141:                                            intSDType, _
1142:                                            boolDecipher
1143:          End If
....:
1147:
....:
....:
....:
```

4.6.5 Connecting to registry keys security descriptor

4.6.5.1 Connecting to registry keys with WMI

The connection to the registry with WMI is possible with the use of the *StdRegProv* class in the **Root\Default** namespace. Although this class exposes the *CheckAccess* method to verify if the user invoking this method possesses some specified permissions, it is not currently possible to retrieve a structural representation of a registry key security descriptor (in the WMI object model, the ADSI object model, or in a binary format). The only way is to use the ADSI **ADsSecurityUtility** or **ADsSecurity** objects, which refer to an ADSI security descriptor access technique.

4.6.5.2 Connecting to registry keys with ADSI

The logic developed for Sample 4.4 ("Connecting to files and folders with ADSI [Part II]") and Sample 4.6 ("Connecting to shares with ADSI [Part IV]") applies for Sample 4.12, since it makes use of the **ADsSecurityUtility** object (Windows Server 2003 or Windows XP only) or the **ADsSecurity** object (Windows 2000 or before). As already mentioned, keep in mind that **ADsSecurity** object does not give access to the SACL component of the security descriptor. The following command line executes the Sample 4.12 code portion:

```
C:\>WMIManageSD.Wsf /RegistryKey:HKLM\SOFTWARE\Microsoft /ADSI+
```

Sample 4.12 *Connecting to registry keys with ADSI (Part X)*

```
....:
....:
....:
1147:
1148:' +-------------------------------------------------------------------------------------------+
1149:' | Registry key                                                                              |
1150:' +-------------------------------------------------------------------------------------------+
1151:         Case cRegistryViaWMI
1152:' WMI technique retrieval ------------------------------------------------------------------
1153:             ' Security descriptor access not implemented via WMI.
1154:
1155:         Case cRegistryViaADSI
1156:' ADSI technique retrieval -------------------------------------------------------------------
1157:             Set objSD = GetSecurityDescriptor (vbNull, _
1158:                                                strRegistryKey, _
1159:                                                intSDType)
1160:             If boolAddAce Then
1161:                Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
1162:                                    objSD, _
1163:                                    strTrustee, _
1164:                                    intACEType, _
1165:                                    intACEMask, _
1166:                                    intACEFlags, _
1167:                                    intACLtype, _
1168:                                    vbNull, _
1169:                                    vbNull, _
1170:                                    intSDType)
1171:             End If
1172:
1173:             If boolDelAce Then
1174:                Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
1175:                                    objSD, _
1176:                                    strTrustee, _
1177:                                    intACLtype, _
1178:                                    intSDType)
1179:             End If
1180:
1181:             If boolOwner Then
1182:                Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
1183:                                       objSD, strOwner, intSDType)
1184:             End If
1185:
1186:             If boolGroup Then
1187:                Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
1188:                                       objSD, strGroup, intSDType)
1189:             End If
1190:
1191:             If boolSDControlFlags Then
1192:                Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
1193:             End If
1194:
1195:             If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
1196:                If boolAddAce Or boolDelAce Then
1197:                   Set objSD = ReOrderACE(vbNull, objSD, intSDType)
1198:                End If
1199:                SetSecurityDescriptor vbNull, _
```

```
1200:                                   objSD, _
1201:                                   strRegistryKey, _
1202:                                   intSDType
1203:             End If
1204:
1205:             If boolViewSD Then
1206:                WScript.Echo
1207:                DecipherADSISecurityDescriptor objSD, _
1208:                                               intSDType, _
1209:                                               boolDecipher
1210:             End If
....:
1213:
....:
....:
....:
```

4.6.6 Connecting to CIM repository namespace security descriptors

4.6.6.1 Connecting to CIM repository namespaces with WMI

With WMI, it is possible to retrieve the security descriptor of a CIM repository namespace (see Sample 4.13). However, the retrieved security descriptor is in a binary form. The GetSecurityDescriptor() function will convert the security descriptor to an ADSI security descriptor representation. With the WMI access method, to get access to the namespace security descriptor, the GetSecurityDescriptor() function requires the **SWBemServices** object created when connecting to the CIM repository namespace (lines 1222 through 1225). It is important to note that the script does not connect to the **Root\CIMv2** namespace anymore, but it connects to the namespace for which the security descriptor must be retrieved. Since the security descriptor is represented in the ADSI object model after its conversion from the binary form, the subsequent subfunctions (AddAce() at line 1232 through 1241, DelAce() at line 1245 through 1249, or ReOrderAce() at line 1270, to name a few) do not require the connection object to the managed entity.

Note that as with a share, the *owner*, *group*, and security descriptor *controls* update is not supported for a CIM repository namespace (lines 1252 through 1266).

Sample 4.13 *Connecting to CIM repository namespaces with WMI (Part XI)*

```
....:
....:
....:
1213:
1214:' +-------------------------------------------------------------------------------------+
1215:' | CIM repository namespace                                                            |
1216:' +-------------------------------------------------------------------------------------+
```

```
1217:         Case cWMINameSpaceViaWMI
1218:' WMI technique retrieval ------------------------------------------------------------------
1219:             objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
1220:             objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
1221:
1222:             Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
1223:                                                              strWMINameSpace, _
1224:                                                              strUserID, _
1225:                                                              strPassword)
....:
1228:             Set objSD = GetSecurityDescriptor (objWMIServices, _
1229:                                                strWMINameSpace, _
1230:                                                intSDType)
1231:             If boolAddAce Then
1232:                Set objSD = AddACE (vbNull, vbNull, vbNull, vbNull, _
1233:                                    objSD, _
1234:                                    strTrustee, _
1235:                                    intACEType, _
1236:                                    intACEMask, _
1237:                                    intACEFlags, _
1238:                                    intACLtype, _
1239:                                    vbNull, _
1240:                                    vbNull, _
1241:                                    intSDType)
1242:             End If
1243:
1244:             If boolDelAce Then
1245:                Set objSD = DelACE (vbNull, vbNull, vbNull, vbNull, _
1246:                                    objSD, _
1247:                                    strTrustee, _
1248:                                    intACLtype, _
1249:                                    intSDType)
1250:             End If
1251:
1252:             ' Not supported for a CIM repository namespace security descriptor.
1253:             '
1254:             ' If boolOwner Then
1255:             '    Set objSD = SetSDOwner(vbNull, vbNull, vbNull, _
1256:             '                           objSD, strOwner, intSDType)
1257:             ' End If
1258:             '
1259:             ' If boolGroup Then
1260:             '    Set objSD = SetSDGroup(vbNull, vbNull, vbNull, _
1261:             '                           objSD, strGroup, intSDType)
1262:             ' End If
1263:             '
1264:             ' If boolSDControlFlags Then
1265:             '    Set objSD = SetSDControlFlags(objSD, intSDControlFlags, intSDType)
1266:             ' End If
1267:
1268:             If boolAddAce Or boolDelAce Or boolOwner Or boolGroup Or boolSDControlFlags Then
1269:                If boolAddAce Or boolDelAce Then
1270:                   Set objSD = ReOrderACE(vbNull, objSD, intSDType)
1271:                End If
1272:
1273:                SetSecurityDescriptor objWMIServices, _
1274:                                      objSD, _
1275:                                      strWMINameSpace, _
1276:                                      intSDType
1277:             End If
```

```
1278:
1279:              If boolViewSD Then
1280:                 WScript.Echo
1281:                 DecipherADSISecurityDescriptor objSD, _
1282:                                                intSDType, _
1283:                                                boolDecipher
1284:              End If
....:
1289:         Case cWMINameSpaceViaADSI
1290:' ADSI technique retrieval ------------------------------------------------------------------
1291:              ' Security descriptor access not implemented via ADSI.
1292:    End Select
1293:
1294:    ]]>
1295:    </script>
1296:  </job>
1297:</package>
```

4.6.6.2 Connecting to CIM repository namespaces with ADSI

Accessing a CIM repository namespace security descriptor is not possible with ADSI and therefore is not supported by the script. The only valid access method is implemented by WMI.

4.7 Accessing the security descriptor set on manageable entities

Now that we have a connection to the manageable entity, we are in a position to retrieve its associated security descriptor. Of course, since the access methods vary, the techniques to read the security descriptor also vary. Moreover, its encoded form, once retrieved, may also vary. This totally relies on the Microsoft ADSI and WMI capabilities. In this section, we will examine the coding techniques used with their conditions of use. The goals of Samples 4.14 through 4.24, which implement the GetSecurityDescriptor() function, are to always return a security descriptor in a manageable form, which means returning an ADSI or a WMI representation of the retrieved security descriptor.

4.7.1 Retrieving file and folder security descriptors

4.7.1.1 Retrieving file and folder security descriptors with WMI

To retrieve the structural representation of a security descriptor from a file or a folder, the *Win32_LogicalFileSecuritySetting GetSecurityDescriptor* method must be used.

It is important to note that the *GetEffectivePermission* method available from the *CIM_DataFile* class (but defined on the parent class, called *CIM_*

LogicalFile) determines whether the caller has the aggregated permissions specified by the permission argument, not only on the file system object but also on the share the file or directory resides on. At no time does this method retrieve a structural representation of the security descriptor.

The use of the *Win32_LogicalFileSecuritySetting GetSecurityDescriptor* method requires a WMI connection to the **Root\CIMv2** namespace. This connection is already executed in Sample 4.3 ("Connecting to files and folders with WMI [Part I]"), which calls the GetSecurityDescriptor() function.

In the case of a WMI security descriptor access method, the parameters of the GetSecurityDescriptor() function represent (Sample 4.14, line 9):

- An **SWBemServices** object representing the WMI connection to the **Root\CIMv2** namespace (contained in the **objConnection** variable).
- A string containing the file or folder path (contained in the **strSource** variable).
- A value representing the security descriptor access method type (contained in the **intSDType** variable), which determines the security descriptor retrieval technique through a **Select Case** statement (line 21). This value is defined in the **SecurityInclude.vbs** included at line 155 of Sample 4.2 ("The **WMIManageSD.Wsf** framework to manage security descriptors from the command line"). The **Select Case** statement determines the execution of lines 26 through 42 for a WMI security descriptor access method for a file or a folder.

With the **SWBemServices** object, the script retrieves an **SWBemObject** object representing an instance of the file specified by the file path name (lines 29 and 30). Next, the script invokes the *GetSecurityDescriptor* method of the *Win32_LogicalFileSecuritySetting* class to retrieve the security descriptor (line 33). Since the retrieved security descriptor is represented in a *Win32_SecurityDescriptor* instance, no further processing is necessary. The GetSecurityDescriptor() function ends its execution by returning the **SWBemObject** object representing the security descriptor instance (line 295).

Sample 4.14 *Retrieving file and folder security descriptors with WMI (Part I)*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function GetSecurityDescriptor (objConnection, strSource, intSDType)
..:
21:    Select Case intSDType
```

```
 22:' +-------------------------------------------------------------------------------------------+
 23:' | File or Folder                                                                            |
 24:' +-------------------------------------------------------------------------------------------+
 25:          Case cFileViaWMI
 26:' WMI retrieval technique ------------------------------------------------------------------
 27:               WScript.Echo "Reading File or Folder security descriptor via WMI from '" & _
 28:                            strSource & "'."
 29:               Set objWMIInstance = objConnection.Get("Win32_LogicalFileSecuritySetting='" & _
 30:                                                      strSource & "'")
 ..:
 33:               intRC = objWMIInstance.GetSecurityDescriptor (objSD)
 34:               If intRC Then
 35:                  WScript.Echo vbCRLF & "Failed to get File or Folder " & _
 36:                               "security descriptor from '" & strSource & "'."
 37:                  WScript.Quit (1)
 38:               End If
 ..:
 42:' Here objSD contains a security descriptor in the WMI object model.
 43:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.1.2 Retrieving file and folder security descriptors with ADSI

To retrieve a file or folder security descriptor with ADSI (see Sample 4.15), we must distinguish if the script runs under Windows Server 2003, Windows XP, Windows 2000, or Windows NT. As mentioned in section 4.6.1.2 ("Connecting to files and folders with ADSI"), under Windows Server 2003 or Windows XP, the script uses the **ADsSecurityUtility** object. The *SecurityMask* property of this object specifies which component of the security descriptor must be retrieved with the *GetSecurityDescriptor* method

Table 4.7 *The ADsSecurityUtility Constants*

Security Descriptor Info	
Name	**Value**
ADS_SECURITY_INFO_OWNER	0x1
ADS_SECURITY_INFO_GROUP	0x2
ADS_SECURITY_INFO_DACL	0x4
ADS_SECURITY_INFO_SACL	0x8
Security Descriptor format	
Name	**Value**
ADS_SD_FORMAT_IID	1
ADS_SD_FORMAT_RAW	2
ADS_SD_FORMAT_HEXSTRING	3
Security Descriptor path	
Name	**Value**
ADS_PATH_FILE	1
ADS_PATH_FILESHARE	2
ADS_PATH_REGISTRY	3

invoked at line 55. The *SecurityMask* property requires a bitwise value, as defined in Table 4.7. By default, only the owner, the group, and the DACL components are retrieved. Because the **ADsSecurityUtility** *GetSecurityDescriptor* method can retrieve a security descriptor from a file, a file system share, or a registry key, this method requires parameters to:

- Specify the path of the entity to retrieve the security descriptor (line 55).
- A constant (as defined in Table 4.7) to determine the nature of the specified path to retrieve (line 56).
- A constant (as defined in Table 4.7) to determine the format of the retrieved security descriptor (line 57).

Sample 4.15 *Retrieving file and folder security descriptors with ADSI (Part II)*

```
..:
..:
..:
43:
44:           Case cFileViaADSI
45:' ADSI retrieval technique ------------------------------------------------------------
46:                 WScript.Echo "Reading File or Folder security descriptor via ADSI from '" & _
47:                              strSource & "'."
48:
49:                 ' Windows Server 2003 only ---------------------------------------------
50:                 objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
51:                                               ADS_SECURITY_INFO_GROUP Or _
52:                                               ADS_SECURITY_INFO_DACL ' Or _
53:                                               ' ADS_SECURITY_INFO_SACL
54:
55:                 Set objSD = objADsSecurity.GetSecurityDescriptor(strSource, _
56:                                                                  ADS_PATH_FILE, _
57:                                                                  ADS_SD_FORMAT_IID)
58:
59:                 ' Windows 2000 only ----------------------------------------------------
60:                 ' Set objSD = objADsSecurity.GetSecurityDescriptor("FILE://" & strSource)
61:
62:                 If Err.Number Then
63:                    WScript.Echo vbCRLF & "Failed to get File or Folder " & _
64:                                 "security descriptor from '" & strSource & "'."
65:                    WScript.Quit (1)
66:                 End If
67:
68:' Here objSD contains a security descriptor in the ADSI object model.
69:
...:
291:           Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

Under Windows 2000 (or Windows NT), the script uses the **ADsSecurity** object implemented by the **ADsSecurity.DLL**. This object also exposes a *GetSecurityDescriptor* method. However, the method requires one single parameter, which is the path of the entity to retrieve the security descriptor. The method uses a "FILE://" pointer to determine that the given path is a file or folder path (line 60). The **ADsSecurity** object does not retrieve the SACL component of a security descriptor and does not expose any *SecurityMask* property (or equivalent) to do so. Because both **ADsSecurityUtility** and **ADsSecurity** objects do not require a connection to the examined file or folder, this parameter is set to Null when calling the GetSecurityDescriptor() function (see Sample 4.4, line 587).

Once completed, the retrieved security descriptor is represented in the ADSI object model, as opposed to the WMI security descriptor access method, which returns a security descriptor in the WMI object model (see Sample 4.14, "Retrieving file and folder security descriptor with WMI [Part I]"). We take advantage of the variant variable type used in scripts to return different types of objects from the same function.

Note: By design, with the **ADsSecurityUtility** object it is possible to retrieve the security descriptor with its SACL component on the condition that the statement of line 53 is specified (ADS_SECURITY_INFO_SACL flag). Another requirement to retrieve the SACL of a security descriptor is that the SE_SECURITY_NAME privilege (also called **SeSecurityPrivilege** or the "Manage auditing and security log" privilege in the GPO) must be granted to the thread/process requesting access to the SACL. Despite the fact that you could run the script as a member of the Administrators group (which has the SE_SECURITY_NAME privilege enabled by default at the account level—see http://msdn.microsoft.com/library/en-us/security/security/sacl_access_right.asp), it is not granted by default at the thread/process level. Therefore, it is necessary for the thread/process to explicity enable this privilege as well. Unfortunately, at writing time, a bug located in the **ADsSecurityUtility** object of Windows Server 2003 and Windows XP does not allow the retrieval of the SACL component, even if the object has been designed for this purpose and even if the statement of line 53 is specified. If you execute the script with the ADS_SECURITY_INFO_SACL flag, the following error message will be returned:

```
A required privilege is not held by the client.
```

Actually, the **ADsSecurityUtility** object retrieving the security descriptor does not grant the ADS_SECURITY_INFO_SACL privilege. Therefore, it

makes it impossible to retrieve the SACL component of the security descriptor.

Due to the timing issues, it is more than likely that Microsoft won't fix this bug for the release code of Windows Server 2003. However, it is possible to use a work-around by developing a small fix. I shared this issue with my colleague, André Larbière from HP Belgium/Luxembourg, and after thinking about different approaches he developed a COM object that could be invoked during the script execution to enable the missing SE_SECURITY_NAME privilege. We won't enter into the details of the COM object development here, but we will see how it can be used from the **WMIManageSD.Wsf** script to work around this problem. Actually, its use is pretty simple. It requires three modifications. The first modification concerns the Sample 4.2 header, where the instantiation of the object enabling the SE_SECURITY_NAME privilege is added at line 195 (**UserRight.Control** object):

```
...:
...:
...:
186:    <reference object="ADs" version="1.0"/>
187:
188:    <!-- ***** Windows Server 2003 only ***** -->
189:    <object progid="ADsSecurityUtility" id="objADsSecurity"/>
190:
191:    <!-- ***** Windows 2000 only ***** -->
192:    <!-- <object progid="ADsSecurity" id="objADsSecurity"/> -->
193:
194:    <object progid="ADSIHelper.SDConversions" id="objADSIHelper"/>
195:    <object progid="UserRight.Control" id="objADSIPriv"/>
196:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
197:    <object progid="WbemScripting.SWbemNamedValueSet" id="objWMINamedValueSet" />
198:
199:    <!-- ***** Windows Server 2003 only ***** -->
200:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
201:
202:    <!-- ***** Windows 2000 only *****-->
203:    <!-- <object progid="SWbemDateTime.WSC" id="objWMIDateTime" /> -->
204:
205:    <script language="VBscript">
206:    <![CDATA[
...:
...:
...:
```

Next, we must make use of the **UserRight.Control** object in the **WMIManageSD.Wsf** script to enable the required privilege. The method exposed by this object to enable the privilege is only used where the **ADsSecurityUtility** object is referenced. This concerns several portions of the script code:

- Sample 4.15 ("Retrieving file and folder security descriptor with ADSI [Part II]") and Sample 4.23 ("Retrieving registry keys security

descriptor with ADSI [Part X]"), both located in the GetSecurityDescriptor() function.

- Sample 4.52 ("Updating file and folder security descriptor with ADSI [Part II]") and Sample 4.60 ("Updating registry keys security descriptor with ADSI [Part X]"), both located in the SetSecurityDescriptor() function.

All samples are subject to the same modification. The following code shows how the script is modified based on Sample 4.15 ("Retrieving file and folder security descriptor with ADSI [Part II]"). The original code is as follows:

```
..:
..:
..:
48:
49:              Windows Server 2003 only ------------------------------------------------
50:             objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
51:                                           ADS_SECURITY_INFO_GROUP Or _
52:                                           ADS_SECURITY_INFO_DACL ' Or _
53:                                           ' ADS_SECURITY_INFO_SACL
54:
55:             Set objSD = objADsSecurity.GetSecurityDescriptor(strSource, _
56:                                                              ADS_PATH_FILE, _
57:                                                              ADS_SD_FORMAT_IID)
58:
..:
..:
..:
```

and must be changed to (lines 54 and 58 in bold):

```
..:
..:
..:
48:
49:             ' Windows Server 2003 only ------------------------------------------------
50:             objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
51:                                           ADS_SECURITY_INFO_GROUP Or _
52:                                           ADS_SECURITY_INFO_DACL ' Or _
53:                                           ADS_SECURITY_INFO_SACL
54:            objADSIPriv.Enable "SeSecurityPrivilege"
55:             Set objSD = objADsSecurity.GetSecurityDescriptor(strSource, _
56:                                                              ADS_PATH_FILE, _
57:                                                              ADS_SD_FORMAT_IID)
58:            objADSIPriv.Disable "SeSecurityPrivilege"
..:
..:
..:
```

Basically, this object is performing the exact same thing as the WMI **wbemPrivilegeSecurity** constant: It grants the SE_SECURITY_NAME privilege (also called **SeSecurityPrivilege**) to the process/thread. Note that

the privilege is enabled before reading the security descriptor and disabled once it has been read.

If you don't plan to use this work-around (i.e., because you don't need to retrieve the SACL), to avoid this problem line 53 must be commented out. In such a case, only the WMI technique is able to retrieve the SACL of a file or folder security descriptor. If you do not need to retrieve the SACL of a file or a folder security descriptor, the native ADSI method is perfectly applicable without this work-around.

4.7.2 Retrieving file system share security descriptors

4.7.2.1 Retrieving file system share security descriptors with WMI

Regarding the file system shares, the *Win32_Share* class exposes (only under Windows Server 2003 or Windows XP) the *GetAccessMask* method to gather information about the share security settings. The *GetAccessMask* method returns the access rights of the share held by the user or group on whose behalf the instance is created. Unfortunately, this method does not return a structural representation of the share security descriptor.

On the other hand, the *Win32_LogicalShareSecuritySetting* class exposes an equivalent method as the *GetSecurityDescriptor* method of the *Win32_LogicalFileSecuritySetting* class. This method retrieves a structural representation of the share security descriptor. Sample 4.16 illustrates this logic. The script retrieves an instance of the *Win32_LogicalShareSecuritySetting* class in an **SWBemObject** object (lines 77 and 78). When a share is created, it is not necessarily created with a security descriptor. In such a case, the share always has a default security setting, even if no security descriptor exists. The default security grants everyone full control. Note that it is the default behavior under Windows NT 4.0 and Windows 2000. However, under Windows XP and Windows Server 2003, the default is everyone read-only. Because the *Win32_LogicalShareSecuritySetting* instance can only be retrieved with the condition that a security descriptor is set, the script manages the situation by testing the error return code of the instance retrieval (lines 79 and 80).

If no instance of this class can be retrieved, it likely means that no specific security is set, which implies the creation of a default security descriptor (line 80) by calling the CreateDefaultSD() function. This function will be examined later in section 4.8 ("Creating a default security descriptor"). If the file system share has a security descriptor set, the script invokes the *GetSecurityDescriptor* method and retrieves the security descriptor in a *Win32_SecurityDescriptor* instance (line 85). The end result is the repre-

sentation of the file system share security descriptor in the WMI object model (line 295).

Sample 4.16 *Retrieving file system share security descriptors with WMI (Part III)*

```
..:
..:
..:
69:
70:' +-------------------------------------------------------------------------------------------+
71:' | Share                                                                                     |
72:' +-------------------------------------------------------------------------------------------+
73:          Case cShareViaWMI
74:' WMI retrieval technique ------------------------------------------------------------------
75:               WScript.Echo "Reading Share security descriptor via WMI from '" & _
76:                            strSource & "'."
77:               Set objWMIInstance = objConnection.Get("Win32_LogicalShareSecuritySetting='" &
78:                                                      strSource & "'")
79:               If Err.Number = wbemErrNotFound Then
80:                  Err.Clear
81:                  Set objSD = CreateDefaultSD (objConnection, intSDType)
82:               Else
83:                  If Err.Number Then ErrorHandler (Err)
84:
85:                  intRC = objWMIInstance.GetSecurityDescriptor (objSD)
86:                  If intRC Then
87:                     WScript.Echo vbCRLF & "Failed to get Share security descriptor from '" &
88:                                  strSource & "'."
89:                     WScript.Quit (1)
90:                  End If
91:               End If
..:
95:' Here objSD contains a security descriptor in the WMI object model.
96:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.2.2 Retrieving file system share security descriptors with ADSI

To retrieve a security descriptor from a file system share via ADSI, the overall logic is basically the same. However, in the details, we must consider some peculiarities:

- It is only possible to retrieve a share security descriptor under Windows Server 2003 (or Windows XP), since these platforms implement the ADSI **ADsSecurityUtility** object.
- Managing the *owner*, the *group*, and the SACL components is not applicable to a share.

Once these restrictions are clearly identified, the script logic is pretty simple (see Sample 4.17). With the *GetSecurityDescriptor* method (line 103) and its correct parameters (as defined in Table 4.7), the script will retrieve the security descriptor in the ADSI object model. If there is no security descriptor set on the share because it uses the default security, then the script creates a default ADSI security descriptor (line 109). Once completed, the GetSecurityDescriptor() function returns the ADSI security descriptor (line 295).

Sample 4.17 *Retrieving file system share security descriptors with ADSI (Part IV)*

```
..:
..:
..:
 96:
 97:           Case cShareViaADSI
 98:' ADSI retrieval technique ------------------------------------------------------------------
 99:                WScript.Echo "Reading Share security descriptor via ADSI from '" & _
100:                             strSource & "'."
101:
102:                ' Windows Server 2003 only ---------------------------------------------
103:                Set objSD = objADsSecurity.GetSecurityDescriptor(strSource, _
104:                                                                 ADS_PATH_FILESHARE, _
105:                                                                 ADS_SD_FORMAT_IID)
106:
107:                If Err.Number Then
108:                   Err.Clear
109:                   Set objSD = CreateDefaultSD (vbNull, intSDType)
110:                End If
111:
112:' Here objSD contains a security descriptor in the ADSI object model.
113:
...:
291:           Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.3 Retrieving Active Directory object security descriptors

4.7.3.1 Retrieving Active Directory object security descriptors with WMI

The access to a security descriptor of an Active Directory object with WMI is a little bit more challenging. The GetSecurityDescriptor() function requires an **SWBemServices** object, which represents a connection to the CIM repository namespace giving access to Active Directory. As we have seen before, this connection is performed in Sample 4.7 ("Connecting to

Active Directory objects with WMI [Part V]"). Next, the **distinguishedName** of the desired object must be correctly parsed. As a reminder, the script command-line parameters executing this portion of the code would be as follows:

```
C:\>WMIManageSD.Wsf /ADObject:"user;CN=MyUser,CN=Users,DC=LissWare,DC=Net"
```

At line 119, the script invokes the ConvertStringInArray() function to split the Active Directory object class from the object **distinguishedName**. Once complete, the returned array contains the Active Directory object class in element 0 and the **distinguishedName** in element 1. With this information, the script retrieves an instance of the specified Active Directory object (lines 132 through 134). For instance, the previous command line will create the following WMI path:

```
ds_user.ADSIPath='LDAP://CN=MyUser,CN=Users,DC=LissWare,DC=Net'")
```

Once the Active Directory object instance is retrieved in an **SWBemObject** object, the script gets access to all attributes of the Active Directory object as represented by WMI. The security descriptor of an Active Directory object is contained in the **nTSecurityDescriptor** attribute. Doing so, the script reads the value of the *DS_nTSecurityDescriptor* property exposed by WMI. Unfortunately, things are not so easy, because the security descriptor contained in this **SWBemObject** object property is in the form of an **SWBemNamedValue** object. This **SWBemNamedValue** object exposes a property called *Value*, which contains the security descriptor in a binary array (line 144). To get the security descriptor in a usable form, the script invokes the *ConvertRawSDToAdsiSD* method exposed by the **ADSIHelper** object (line 144). The **ADSIHelper** object is instantiated in Sample 4.2 at line 194. We will come back to the **ADSIHelper** object in section 4.9 ("The security descriptor conversion"). The end result is a security descriptor represented in the ADSI object model. In this particular case, we have a mixed situation, since the script uses a WMI access method, which, in the end, returns an ADSI security descriptor (line 295).

Sample 4.18 *Retrieving Active Directory object security descriptors with WMI (Part V)*

```
...:
...:
...:
113:
114:' +-------------------------------------------------------------------------------------+
115:' | Active Directory object                                                             |
116:' +-------------------------------------------------------------------------------------+
117:          Case cActiveDirectoryViaWMI
118:' WMI retrieval technique ---------------------------------------------------------------
119:               arrayADInfo = ConvertStringInArray (strSource, ";")
```

```
120:                If Ubound(arrayADInfo) <> 1 Then
121:                   WScript.Echo "The Active Directory class and object distinguished name " & _
122:                                "must be specified as follow:" & vbCRLF & vbCRLF & _
123:                                "/ADObject:""Objectclass;CN=MyObject," & _
124:                                "CN=Users,DC=LissWare,DC=Net"""
125:                   WScript.Quit (1)
126:                End If
127:
128:                WScript.Echo "Reading " & LCase(arrayADInfo(0)) & " Active Directory " & _
129:                             "object security descriptor via WMI from 'LDAP://" & _
130:                             arrayADInfo(1) & "'."
131:
132:                Set objWMIInstance = objConnection.Get("ds_" & LCase(arrayADInfo(0)) & _
133:                                                       ".ADSIPath='LDAP://" & _
134:                                                       arrayADInfo(1) & "'")
...:
137:                Set objSD = objWMIInstance.DS_nTSecurityDescriptor
138:                If Err.Number Then
139:                   Err.Clear
140:                   WScript.Echo vbCRLF & "Failed to get Active Directory object " & _
141:                                "security descriptor from 'LDAP://" & strSource & "'."
142:                   WScript.Quit (1)
143:                Else
144:                   Set objSD = objADSIHelper.ConvertRawSDToAdsiSD (objSD.Value)
145:                   If Err.Number Then ErrorHandler (Err)
146:                End If
...:
150:' Here objSD contains a security descriptor in the ADSI object model.
151:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.3.2 Retrieving Active Directory object security descriptors with ADSI

Retrieving a security descriptor from an Active Directory object with ADSI is, of course, a more natural technique, since ADSI is especially designed for Active Directory accesses. Despite this fact, there are some things we must still take into consideration. Sample 4.19 illustrates this technique. Although the script uses ADSI, it is required to have a connection to the Active Directory object (called an object binding in the LDAP world). This object binding is established in Sample 4.8 ("Connecting to Active Directory objects with ADSI [Part VI]") and passed in the **objConnection** variable to the GetSecurityDescriptor() function. Basically, this object represents the Active Directory object in the ADSI object model. From this object, the script retrieves the **ntSecurityDescriptor** attribute (line 161). To retrieve the SACL component, the script must initialize the *SetOption* property of this object (lines 156 through 159). The *SetOption* property works

in the same way as the *SecurityMask* property of the **ADsSecurityUtility** object. There is no difference between the two properties, but the *SetOption* property is exposed by the ADSI representation of the Active Directory object. Once completed, the GetSecurityDescriptor() function returns an ADSI representation of the Active Directory object security descriptor (line 295).

Sample 4.19 *Retrieving Active Directory object security descriptors with ADSI (Part VI)*

```
...:
...:
...:
151:
152:          Case cActiveDirectoryViaADSI
153:' ADSI retrieval technique ------------------------------------------------------------------
154:               WScript.Echo "Reading Active Directory object security descriptor " & _
155:                            "via ADSI from 'LDAP://" & strSource & "'."
156:               objConnection.SetOption ADS_OPTION_SECURITY_MASK, ADS_SECURITY_INFO_OWNER Or _
157:                                                                 ADS_SECURITY_INFO_GROUP Or _
158:                                                                 ADS_SECURITY_INFO_DACL Or _
159:                                                                 ADS_SECURITY_INFO_SACL
160:
161:               Set objSD = objConnection.Get("ntSecurityDescriptor")
162:               If Err.Number Then
163:                  Err.Clear
164:                  WScript.Echo vbCRLF & "Failed to get Active Directory object " & _
165:                               "security descriptor from 'LDAP://" & strSource & "'."
166:                  WScript.Quit (1)
167:               End If
168:
169:' Here objSD contains a security descriptor in the ADSI object model.
170:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.4 Retrieving Exchange 2000 mailbox security descriptors

4.7.4.1 Retrieving Exchange 2000 mailbox security descriptors with WMI

Retrieving the Exchange 2000 mailbox security descriptor with WMI (Sample 4.20) follows the same logic as retrieving an Active Directory object security descriptor (Sample 4.18). However, only **user** objects are accessed, and, instead of reading the *DS_nTSecurityDescriptor* property of the **SWBemObject** object representing the Active Directory object instance, the script must read the *DS_msExchMailboxSecurityDescriptor*

property (line 182). As in Sample 4.18 ("Retrieving Active Directory object security descriptors with WMI [Part V]"), the security descriptor is finally retrieved in a binary array and must be converted with the help of the *ConvertRawSDToAdsiSD* method exposed by the **ADSIHelper** object (line 189). Once completed, the GetSecurityDescriptor() function returns an ADSI security descriptor representation accessed by WMI (line 295).

Sample 4.20 *Retrieving Exchange 2000 mailbox security descriptors with WMI (Part VII)*

```
...:
...:
...:
170:
171:' +-------------------------------------------------------------------------------------+
172:' | Exchange 2000 mailbox                                                               |
173:' +-------------------------------------------------------------------------------------+
174:          Case cExchange2000MailboxViaWMI
175:' WMI retrieval technique ------------------------------------------------------------
176:               WScript.Echo "Reading Exchange 2000 mailbox security descriptor " & _
177:                            "via WMI from 'LDAP://" & strSource & "'."
178:               Set objWMIInstance = objConnection.Get("ds_user.ADSIPath='LDAP://" & _
179:                                                      strSource & "'")
...:
182:               Set objSD = objWMIInstance.DS_msExchMailboxSecurityDescriptor
183:               If Err.Number Then
184:                  Err.Clear
185:                  WScript.Echo vbCRLF & "Failed to get Exchange 2000 mailbox " & _
186:                                        "security descriptor from 'LDAP://" & strSource & "'."
187:                  WScript.Quit (1)
188:               Else
189:                  Set objSD = objADSIHelper.ConvertRawSDToAdsiSD (objSD.Value)
190:                  If Err.Number Then ErrorHandler (Err)
191:               End If
...:
195:' Here objSD contains a security descriptor in the ADSI object model.
196:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.4.2 Retrieving Exchange 2000 mailbox security descriptors with ADSI

Retrieving an Exchange 2000 mailbox security descriptor with ADSI is as simple as retrieving an Active Directory object security descriptor. Instead of accessing the **ntSecurityDescriptor** attribute, the script must retrieve the **msExchMailboxSecurityDescriptor** attribute (Sample 4.21, line 206). To ensure that all components of the security descriptor are retrieved, the script

initializes the *SetOption* property of the Active Directory object (lines 201 through 204).

Sample 4.21 *Retrieving Exchange 2000 mailbox security descriptors with ADSI (Part VIII)*

```
...:
...:
...:
196:
197:          Case cExchange2000MailboxViaADSI
198:' ADSI retrieval technique ------------------------------------------------------------------
199:               WScript.Echo "Reading Exchange 2000 mailbox security descriptor " & _
200:                            "via ADSI from 'LDAP://" & strSource & "'."
201:               objConnection.SetOption ADS_OPTION_SECURITY_MASK, ADS_SECURITY_INFO_OWNER Or _
202:                                                                 ADS_SECURITY_INFO_GROUP Or _
203:                                                                 ADS_SECURITY_INFO_DACL Or _
204:                                                                 ADS_SECURITY_INFO_SACL
205:
206:               Set objSD = objConnection.Get("msExchMailboxSecurityDescriptor")
207:               If Err.Number Then
208:                  Err.Clear
209:                  WScript.Echo vbCRLF & "Failed to get Exchange 2000 mailbox " & _
210:                               "security descriptor from 'LDAP://" & strSource & "'."
211:                  WScript.Quit (1)
212:               End If
213:
214:' Here objSD contains a security descriptor in the ADSI object model.
215:
...:
291:          Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.4.3 Retrieving Exchange 2000 mailbox security descriptors with CDOEXM

If Exchange 2000 Service Pack 2 is installed, it is possible to retrieve the Exchange 2000 mailbox security descriptors with CDOEXM. Since CDOEXM acts as an extension for ADSI, the coding technique is very similar to the ADSI coding in Sample 4.21. However, instead of accessing the security descriptor stored in the Active Directory, the CDOEXM method accesses the security descriptor stored in the Exchange store. Actually, both security descriptors are the same, but the Active Directory security descriptor is the mirror of the one in the Exchange store. This will be very important when updating the Exchange 2000 mailbox security descriptors (see section 4.13.4.3, "Updating Exchange 2000 mailbox security descriptors with CDOEXM").

To get the mailbox security descriptor, the script reads the *MailboxRights* property exposed by the **IExchangeMailbox** CDOEXM interface (line 220). The end result is a security descriptor in the ADSI object model (line 295).

Sample 4.22 *Retrieving Exchange 2000 mailbox security descriptors with CDOEXM (Part IX)*

```
...:
...:
...:
215:
216:           Case cExchange2000MailboxViaCDOEXM
217:' CDOEXM retrieval technique ------------------------------------------------------------------
218:                WScript.Echo "Reading Exchange 2000 mailbox security descriptor " & _
219:                             "via CDOEXM from 'LDAP://" & strSource & "'."
220:                Set objSD = objConnection.MailboxRights
221:                If Err.Number Then
222:                   Err.Clear
223:                   WScript.Echo vbCRLF & "Failed to get Exchange 2000 mailbox " & _
224:                                "security descriptor from 'LDAP://" & strSource & "'."
225:                   WScript.Quit (1)
226:                End If
227:
228:' Here objSD contains a security descriptor in the ADSI object model.
229:
...:
291:           Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.7.5 Retrieving registry key security descriptors with ADSI

In Chapter 3 (section 3.3.3, "Registry providers"), we saw that WMI implements a WMI *Registry* provider to support the *StdRegProv* class. Unfortunately, this class does not implement a method to retrieve a structural representation of a security descriptor. The only method that gives information about the registry key security is the *CheckAccess* method. This method verifies that the user invoking the method possesses the specified permissions. Although useful, this WMI method does not suit the needs of the **WMIManageSD.Wsf** script. We must use an alternate tactic implemented by ADSI to retrieve the desired information.

To retrieve a registry key security descriptor with ADSI (see Sample 4.23), we must determine if the script runs under Windows XP, Windows Server 2003, Windows 2000, or Windows NT. The selected platform deter-

mines the use of the **ADsSecurityUtility** object of Windows XP and Windows Server 2003 (lines 243 through 251) or the **ADsSecurity** object of Windows 2000 and Windows NT 4.0 (line 254). Under Windows Server 2003, to retrieve the SACL component of the security descriptor, the *SecurityMask* property of the **ADsSecurityUtility** must be initialized (lines 244 through 247). Once complete, the script retrieves the registry key security descriptor by invoking the *GetSecurityDescriptor* method of the **ADsSecurityUtility** object (lines 249 through 251). The method uses the required parameters, as specified in Table 4.7 ("The **ADsSecurityUtility** Constants").

Sample 4.23 *Retrieving registry key security descriptors with ADSI (Part X)*

```
...:
...:
...:
229:
230:' +-------------------------------------------------------------------------------------+
231:' | Registry key                                                                        |
232:' +-------------------------------------------------------------------------------------+
233:          Case cRegistryViaWMI
234:' WMI retrieval technique ------------------------------------------------------------
235:
236:' Here we can't retrieve a security descriptor via this access method.
237:
238:          Case cRegistryViaADSI
239:' ADSI retrieval technique -----------------------------------------------------------
240:               WScript.Echo "Reading registry security descriptor " & _
241:                            "via ADSI from '" & strSource & "'."
242:
243:               ' Windows Server 2003 only  ---------------------------------------------
244:               objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
245:                                             ADS_SECURITY_INFO_GROUP Or _
246:                                             ADS_SECURITY_INFO_DACL ' Or _
247:                                             ' ADS_SECURITY_INFO_SACL
248:
249:               Set objSD = objADsSecurity.GetSecurityDescriptor(strSource, _
250:                                                                ADS_PATH_REGISTRY, _
251:                                                                ADS_SD_FORMAT_IID)
252:
253:               ' Windows 2000 only -----------------------------------------------------
254:               ' Set objSD = objADsSecurity.GetSecurityDescriptor("RGY://" & strSource)
255:
256:               If Err.Number Then
257:                  WScript.Echo vbCRLF & "Failed to get registry security descriptor " & _
258:                               "from '" & strSource & "'."
259:                  WScript.Quit (1)
260:               End If
261:
262:' Here objSD contains a security descriptor in the ADSI object model.
263:
...:
291:          Case Else
```

```
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

Since the **ADsSecurityUtility** object is subject to a bug under Windows Server 2003 and Windows XP, don't forget to refer to section 4.7.1.2 ("Retrieving file and folder security descriptors with ADSI"). If the workaround explained in that section is not used, and since the WMI technique is not able to retrieve a structural representation of a registry key security decriptor, there will be no way to retrieve the SACL of registry key security descriptors from the scripting world with the current ADSI COM objects.

Under Windows 2000 (or Windows NT), the *GetSecurityDescriptor* method of the **ADsSecurity** object is invoked (line 254). As mentioned before, no SACL component can be retrieved with this object. Once finished, the GetSecurityDescriptor() function returns the registry key security descriptor in the ADSI object model (line 295).

4.7.6 Retrieving CIM repository namespace security descriptors with WMI

To retrieve the security descriptor of a CIM repository namespace, the script must refer to a WMI system class called *__SystemSecurity*. As usual, to get access to an instance of a class, the script must connect to its corresponding CIM repository namespace. In this case, the script will be connected to the namespace from which we must retrieve the security descriptor. This connection is made in Sample 4.13 ("Connecting to CIM repository namespaces with WMI [Part XI]"), which provides an **SWBemServices** object to the GetSecurityDescriptor() function. From this connection, Sample 4.24 retrieves an instance of the *__SystemSecurity* class (line 271). Note that the *__SystemSecurity* class is a singleton class, which makes sense, since WMI implements one security descriptor per namespace. To get the namespace security descriptor, the *__SystemSecurity* class exposes the *GetSD* method (line 274). Unfortunately, things are not straightforward. The retrieved security descriptor has the format of a binary array, which forces the script to convert it to an ADSI security descriptor with the help of the **ADSIHelper** object (line 280). As a reminder, the **ADSIHelper** object is instantiated in Sample 4.2 at line 194. We will come back to the **ADSIHelper** object in section 4.9 ("The security descriptor conversion"). Once completed, the GetSecurityDescriptor() function returns a CIM

repository namespace security descriptor retrieved with WMI but formatted as an ADSI security descriptor (line 295).

Sample 4.24 *Retrieving CIM repository namespace security descriptors with WMI (Part XI)*

```
...:
...:
...:
263:
264:' +-------------------------------------------------------------------------------------+
265:' | CIM repository namespace                                                            |
266:' +-------------------------------------------------------------------------------------+
267:           Case cWMINameSpaceViaWMI
268:' WMI retrieval technique ------------------------------------------------------------
269:               WScript.Echo "Reading CIM repository namespace security descriptor via WMI from '" & _
270:                            strSource & "'."
271:                Set objWMIInstance = objConnection.Get("__SystemSecurity=@")
...:
274:                intRC = objWMIInstance.GetSD (arrayBytes)
275:                If intRC Then
276:                   WScript.Echo vbCRLF & "Failed to get CIM repository namespace security " & _
277:                                "descriptor from '" & strSource & "'."
278:                   WScript.Quit (1)
279:                Else
280:                   Set objSD = objADSIHelper.ConvertRawSDToAdsiSD (arrayBytes)
281:                   If Err.Number Then ErrorHandler (Err)
282:                End If
283:
284:' Here objSD contains a security descriptor in the ADSI object model.
285:
286:           Case cWMINameSpaceViaADSI
287:' ADSI retrieval technique ---------------------------------------------------------------
288:
289:' Here we can't retrieve a security descriptor via this access method.
290:
291:           Case Else
292:
293:    End Select
294:
295:    Set GetSecurityDescriptor = objSD
...:
299:End Function
```

4.8 Creating a default security descriptor

In Sample 4.16 ("Retrieving file system share security descriptors with WMI [Part III]") and Sample 4.17 ("Retrieving file system share security descriptors with ADSI [Part IV]"), we saw that it was necessary to create a default security descriptor by invoking the CreateDefaultSD() function (Sample 4.25). This is necessary because a file system share does not necessarily have a security descriptor defined. Because the file system share has a default behavior when no security descriptor is set, the script provides the

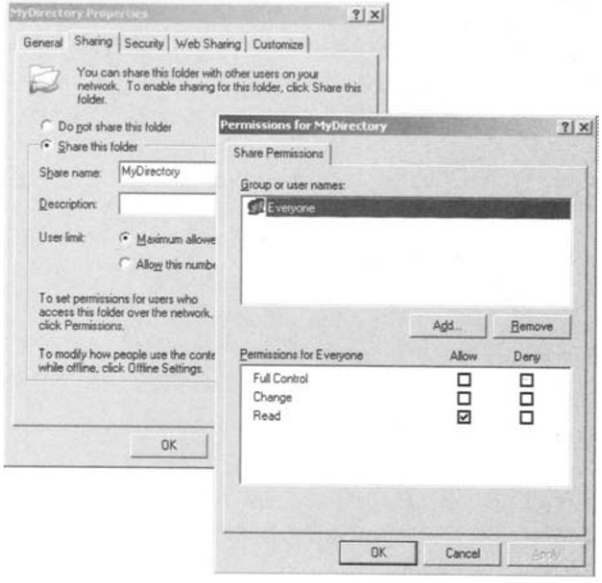


Figure 4.17
The default share security descriptor.

exact same default security descriptor as the one supposed to be present. So, when examining a share, this avoids any confusion for situations where no security descriptor is set for a share. Windows Explorer behaves in the same way, since it shows a security descriptor set on the share (Figure 4.17). If a script tries to retrieve the security descriptor, the code will return an error stating that no security descriptor is available. Therefore, Sample 4.25 creates the corresponding security descriptor.

It is important to note that the file system share is the only situation where a default security descriptor is created if it is missing. This is the reason why Sample 4.25 creates a default security descriptor only for file system share security descriptors accessed with WMI (lines 36 through 58) or ADSI (lines 60 through 84). All other situations expect to find a security descriptor. If there is no security descriptor set with all other manageable entities, then the script will return an error claiming that no security descriptor is available. This situation is shown in Samples 4.14 through 4.24.

As shown in Sample 4.25 (lines 38 through 58), when the security descriptor must be returned in the WMI object model, the scripts uses the **SWBemServices** object to create three new instances: one instance from the *Win32_SecurityDescriptor* class (line 38), one from the *Win32_ACE* class (line 39), and one instance from the *Win32_Trustee* class (line 40). These three new instances are the required instances to create a new security descriptor in the WMI object model. However, creating these instances is

not enough. Each of them must be properly initialized. Sample 4.25 executes this task from line 42 through 58 for the WMI object model. From line 42 through 46, the script creates the trustee for the "Everyone" group. The trustee creation requires the SID of the group. Because the "Everyone" group is a built-in group available from all Windows installations, it can be created immediately with its very well known SID (S-1-1-0) in its string representation (line 44) and in its binary form (line 45).

Sample 4.25 *Create a default security descriptor for a share*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function CreateDefaultSD (objWMIServices, intSDType)
..:
18:    Select Case intSDType
..:
22:           Case cFileViaWMI
..:
27:           Case cFileViaADSI
..:
32:' +-------------------------------------------------------------------------------------+
33:' | Share                                                                               |
34:' +-------------------------------------------------------------------------------------+
35:           Case cShareViaWMI
36:' WMI creation technique ---------------------------------------------------------------
37:
38:               Set objNewSD = objWMIServices.Get("Win32_SecurityDescriptor").SpawnInstance_()
39:               Set objNewACE = objWMIServices.Get("Win32_ACE").SpawnInstance_()
40:               Set objNewTrustee = objWMIServices.Get("Win32_Trustee").SpawnInstance_()
41:
42:               objNewTrustee.Domain = Null
43:               objNewTrustee.Name = "Everyone"
44:               objNewTrustee.SIDString = "S-1-1-0"
45:               objNewTrustee.SID = Array (1,1,0,0,0,0,0,1,0,0,0,0)
46:               objNewTrustee.SidLength = 12
47:
48:               objNewACE.Trustee = objNewTrustee
49:               objNewACE.AceType = ACCESS_ALLOWED_ACE_TYPE
50:               objNewACE.AccessMask = FILE_SHARE_FULL_ACCESS Or _
51:                                      FILE_SHARE_CHANGE_ACCESS Or _
52:                                      FILE_SHARE_READ_ACCESS
53:               objNewACE.AceFlags = 0
54:
55:               objNewSD.DACL = Array (objNewAce)
56:               objNewSD.ControlFlags = SE_SELF_RELATIVE Or SE_DACL_PRESENT
57:
58:               ' Here objNewSD contains a security descriptor in the WMI object model.
59:
60:           Case cShareViaADSI
61:' ADSI creation technique ----------------------------------------------------------------
62:
63:               ' Windows Server 2003 only
64:               Set objNewSD = CreateObject ("SecurityDescriptor")
65:               Set objNewACL = CreateObject ("AccessControlList")
```

```
 66:                Set objNewACE = CreateObject ("AccessControlEntry")
 67:
 68:                objNewACE.Trustee = "Everyone"
 69:                objNewACE.AceType = ACCESS_ALLOWED_ACE_TYPE
 70:                objNewACE.AccessMask = FILE_SHARE_FULL_ACCESS Or _
 71:                                       FILE_SHARE_CHANGE_ACCESS Or _
 72:                                       FILE_SHARE_READ_ACCESS
 73:                objNewACE.AceFlags = 0
 74:
 75:                objNewACL.AddAce objNewAce
 76:
 77:                objNewSD.DiscretionaryACL = objNewACL
 78:                objNewSD.Revision = 1
 79:                objNewSD.Control = SE_SELF_RELATIVE Or SE_DACL_PRESENT
 ..:
 84:                ' Here objNewSD contains a security descriptor in the ADSI object model.
 ..:
 94:            Case cActiveDirectoryViaADSI
 ..:
102:            Case cExchange2000MailboxViaWMI
...:
107:            Case cExchange2000MailboxViaADSI
...:
112:            Case cExchange2000MailboxViaCDOEXM
...:
120:            Case cRegistryViaWMI
...:
125:            Case cRegistryViaADSI
...:
133:            Case cWMINameSpaceViaWMI
...:
138:            Case cWMINameSpaceViaADSI
...:
145:    End Select
146:
147:    Set CreateDefaultSD = objNewSD
...:
151:End Function
```

Next, the script initializes the rights that are necessary to grant full control access to the "Everyone" group (lines 48 through 58). Note that under Windows XP and Windows Server 2003, the default access on a share for the "Everyone" group is read-only (FILE_SHARE_READ_ACCESS flag). These values are defined in the **SecurityInclude.vbs** included at line 155 in Sample 4.2 ("The **WMIManageSD.Wsf** framework to manage security descriptors from the command line"). We will see later, in section 4.11.4 ("Deciphering the Access Control Entries"), how to select the values to create some specific rights. Once complete, the CreateDefaultSD() function returns a WMI security descriptor representation of the rights set on a file system share, as shown in Figure 4.17.

When the security descriptor must be returned in the ADSI object model, the script follows the same logic. However, it uses the ADSI object model, which means that it creates the ADSI objects representing an ADSI

security descriptor (lines 64 through 66). So, it creates a security descriptor (line 64), an Access Control List (line 65), and an Access Control Entry (line 66). Once created, the script initializes the various values to grant the required right to the "Everyone" group, as shown in Figure 4.17.

An important point to note here is about the values defining the rights. Even if the security descriptor can be represented in the ADSI object model or in the WMI object model, it is interesting to see that the assigned values are always the same. This means that the values used to decipher a security descriptor remain a constant, independent of the object model used. This is an important point to remember when we decipher the security descriptor in section 4.10 ("Deciphering the security descriptor").

4.9 The security descriptor conversion

We have seen situations where the security descriptor is not directly retrieved in the WMI or in the ADSI object model. Sometimes, the security descriptor is represented as a binary array, which does not represent a usable form from a script. In such a case, it is necessary to convert the binary security descriptor to a usable object model. We mentioned that ADSI represents a good alternative to convert the binary security descriptor into the ADSI object model compared with the low-level programming that Win32 APIs require. Moreover, this approach is much simpler than digging into the low-level API programming. As a reminder, this conversion to get an ADSI security descriptor is necessary for:

- Sample 4.18, "Retrieving Active Directory object security descriptors with WMI (Part V)"
- Sample 4.20, "Retrieving Exchange 2000 mailbox security descriptors with WMI (Part VII)"
- Sample 4.24, "Retrieving CIM repository namespace security descriptors with WMI (Part XI)"

We will see further that a conversion is also necessary to update the security descriptor for:

- Sample 4.55, "Updating Active Directory object security descriptors with WMI (Part V)"
- Sample 4.57, "Updating Exchange 2000 mailbox security descriptors with WMI (Part VII)"
- Sample 4.61, "Updating CIM repository namespace security descriptors with WMI (Part XI)"

For the variable type conversion facility, the security descriptor conversion is encapsulated into a COM object called **ADSIHelper**. This ActiveX DLL object is written in Visual BASIC v6.0 (VB6), which allows the use of type definitions for variables. This comes from the fact that the ADSI technique used for the conversion requires a precise type as input parameter. Since scripting languages do not have explicit type definitions (Variant), this makes VB6 a good candidate for such programming while maintaining simplicity of use. The **ADSIHelper** object exposes two methods:

- *ConvertRawSDToAdsiSD*, which converts the binary security descriptor to an ADSI representation. The code is shown in Sample 4.26.
- *ConvertAdsiSDToRawSD*, which converts the ADSI security descriptor to a binary format. The code is shown in Sample 4.27.

Here again, we must determine if we are running under Windows Server 2003 (or Windows XP) or Windows 2000 (or Windows NT). Under Windows Server 2003 (or Windows XP), the **ADSIHelper** ActiveX DLL uses the **ADsSecurityUtility** object to convert the security descriptor (lines 25 through 33). It uses the *ConvertSecurityDescriptor* method exposed by the **ADsSecurityUtility** object. This method can convert a binary security descriptor to an ADSI security descriptor and vice versa (Sample 4.26, lines 31 through 33, and Sample 4.27, lines 72 through 74). The method uses the values defined in Table 4.7 ("The **ADsSecurityUtility** constants"). Because the values required by these conversion techniques must use a formal type, it makes it easier to program this logic under VB6 than a script, which only uses the Variant type.

Sample 4.26 *Converting the binary security descriptor to an ADSI representation*

```
 1:Option Explicit
 2:
 3:Public Function ConvertRawSDToAdsiSD(ByVal arrayBytes As Variant) As IADsSecurityDescriptor
 4:
 5:' Windows Server 2003 only
 6:Dim objADsSecurity As New ADsSecurityUtility
 7:
 8:' Windows 2000 only
 9:' Dim objPropertyValue As New PropertyValue
10:
11:Dim objADsSD As IADsSecurityDescriptor
12:Dim byteSD() As Byte
13:Dim intIndice As Integer
14:
15:    On Error Resume Next
16:
17:    ' We must convert the variant array to a byte array.
18:    ' Otherwise, doesn't work ... (type conversion problem)
19:    ReDim byteSD(UBound(arrayBytes))
```

```
20:
21:     For intIndice = 0 To UBound(arrayBytes)
22:         byteSD(intIndice) = arrayBytes(intIndice)
23:     Next
24:
25:       Windows Server 2003 only ------------------------------------------------------
26:     objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
27:                                   ADS_SECURITY_INFO_GROUP Or _
28:                                   ADS_SECURITY_INFO_DACL Or _
29:                                   ADS_SECURITY_INFO_SACL
30:
31:     Set objADsSD = objADsSecurity.ConvertSecurityDescriptor(byteSD, _
32:                                                             ADS_SD_FORMAT_RAW, _
33:                                                             ADS_SD_FORMAT_IID)
34:
35:     ' Windows 2000 only -----------------------------------------------------------
36:     ' objPropertyValue.PutObjectProperty ADSTYPE_OCTET_STRING, (byteSD)
37:     ' Set objADsSD = objPropertyValue.GetObjectProperty(ADSTYPE_NT_SECURITY_DESCRIPTOR)
38:
39:     If Err.Number Then
40:        ErrorHandler "ConvertRawSDToAdsiSD", _
41:                     "Unable to convert raw Security Descriptor to an ADSI Security Descriptor",
42:                     Err, _
43:                     True
..:
45:        Exit Function
46:     End If
47:
48:     Set ConvertRawSDToAdsiSD = objADsSD
49:
50:End Function
51:
..:
..:
..:
```

Under Windows 2000 (or Windows NT), the **ADSIHelper** ActiveX DLL uses the **PropertyValue** ADSI object exposed by the ADSI **IADsPropertyValue** interface (lines 36 and 37). The ActiveX DLL creates the **PropertyValue** object (line 9) and uses its *PutObjectProperty* method to assign the binary security descriptor as value (line 36). With the help of the ADSI cache and the ADSI property conversion mechanisms, the ActiveX DLL reads the assigned value by requesting an ADSI security descriptor format (line 37). The end result is that an ADSI security descriptor is returned from the ActiveX DLL. Note that this technique is also applicable under Windows Server 2003 (or Windows XP). Here, the **ADsSecurityUtility** object is only used to demonstrate some of the new ADSI capabilities provided by Windows Server 2003 and Windows XP.

The conversion of an ADSI security descriptor to a binary security descriptor follows the exact same logic (see Sample 4.27). Only the conversion types requested are inverted when compared with Sample 4.26.

Sample 4.27 *Converting the ADSI security descriptor to a binary format*

```
..:
..:
..:
51:
52:Public Function ConvertAdsiSDToRawSD(objVariantADsSD As Variant) As Variant()
53:
54:' Windows Server 2003 only
55:Dim objADsSecurity As New ADsSecurityUtility
56:
57:' Windows 2000 only
58:' Dim objPropertyValue As New PropertyValue
59:
60:Dim arrayBytes() As Variant
61:Dim byteSD() As Byte
62:Dim intIndice As Integer
63:
64:    On Error Resume Next
65:
66:    ' Windows Server 2003 only ------------------------------------------------------
67:    objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
68:                                  ADS_SECURITY_INFO_GROUP Or _
69:                                  ADS_SECURITY_INFO_DACL Or _
70:                                  ADS_SECURITY_INFO_SACL
71:
72:    byteSD = objADsSecurity.ConvertSecurityDescriptor(objVariantADsSD, _
73:                                                      ADS_SD_FORMAT_IID, _
74:                                                      ADS_SD_FORMAT_RAW)
75:
76:    ' Windows 2000 only ------------------------------------------------------------
77:    ' objPropertyValue.PutObjectProperty ADSTYPE_NT_SECURITY_DESCRIPTOR, (objVariantADsSD)
78:    ' byteSD = objPropertyValue.GetObjectProperty(ADSTYPE_OCTET_STRING)
79:
80:    If Err.Number Then
81:       ErrorHandler "ConvertAdsiSDToRawSD", _
82:                    "Unable to convert ADSI Security Descriptor to a raw Security Descriptor",
83:                    Err, _
84:                    True
85:       ConvertAdsiSDToRawSD = Null
86:       Exit Function
87:    End If
88:
89:    ' We must convert the byte array to a variant array.
90:    ' Otherwise, doesn't work ... (type conversion problem)
91:    ReDim arrayBytes(UBound(byteSD))
92:
93:    For intIndice = 0 To UBound(byteSD)
94:        arrayBytes(intIndice) = byteSD(intIndice)
95:    Next
96:
97:    ConvertAdsiSDToRawSD = arrayBytes
98:
99:End Function
100:
...:
...:
...:
```

4.10 Deciphering the security descriptor

To decipher the security descriptor, it is important to consider the object model used for its representation, because the object model organization will heavily influence the algorithm. As we have seen, sometimes we have a security descriptor represented in the WMI object model; sometimes we have a security descriptor represented in the ADSI object model. This implies that both object models referenced from Samples 4.3 through 4.13 must be handled in the script. Actually, the following portions of the code use a WMI deciphering technique:

- Sample 4.3, "Connecting to files and folders with WMI (Part I)"
- Sample 4.5, "Connecting to shares with WMI (Part III)"

All other portions of the code use an ADSI deciphering technique (see Sample 4.29):

- Sample 4.4, "Connecting to files and folders with ADSI (Part II)"
- Sample 4.6, "Connecting to shares with ADSI (Part IV)"
- Sample 4.7, "Connecting to Active Directory objects with WMI (Part V)"
- Sample 4.8, "Connecting to Active Directory objects with ADSI (Part VI)"
- Sample 4.9, "Connecting to Exchange 2000 mailbox information with WMI (Part VII)"
- Sample 4.10, "Connecting to Exchange 2000 mailbox information with ADSI (Part VIII)"
- Sample 4.11, "Connecting to Exchange 2000 mailbox information with CDOEXM (Part IX)"
- Sample 4.12, "Connecting to registry keys with ADSI (Part X)"
- Sample 4.13, "Connecting to CIM repository namespaces with WMI (Part XI)"

4.10.1 Deciphering the WMI security descriptor representation

Let's start with the WMI deciphering technique. Sample 4.28 shows the logic used to decipher a WMI security descriptor representation. As we have seen at the beginning of this chapter, in Figure 4.11, the *Win32_Security-*

Descriptor instance contains some *Win32_Trustee* instances (i.e., the group and the owner component) and usually one or two collections of *Win32_ACE* instances (i.e., DACL and SACL components), which in turn contain other *Win32_Trustee* instances (i.e., Trustee). With this peculiarity and the information about the nature of a WMI property (CIM type), it is possible to make use of a recursive algorithm. Each time the code encounters an object instance when it examines an instance property, the routine will call itself to decipher the instance.

Sample 4.28 *Deciphering a WMI security descriptor representation*

```
.:
.:
.:
8:' ------------------------------------------------------------------------------------------
9:Function DecipherWMISecurityDescriptor (objWMIInstance, _
10:                                       intSDType, _
11:                                       ByVal strIndent, _
12:                                       boolDecipher)
..:
19:    WScript.Echo strIndent & "+- " & _
20:                 objWMIInstance.Path_.Class & " " & _
21:                 String (90 - Len (objWMIInstance.Path_.Class) - Len (strIndent), "-")
22:
23:    Set objWMIPropertySet = objWMIInstance.Properties_
24:    For Each objWMIProperty In objWMIPropertySet
25:        If Not IsNull (objWMIProperty.Value) Then
26:           If objWMIProperty.CIMType = wbemCimtypeObject Then
27:              If objWMIProperty.IsArray Then
28:
29:                 ' This is an array, we deal with the Win32_ACE
30:                 DisplayFormattedProperty Null, _
31:                                          strIndent & "| " & objWMIProperty.Name, _
32:                                          "(Win32_ACE)", _
33:                                          Null
34:                 For Each varElement In objWMIProperty.Value
35:                     DecipherWMISecurityDescriptor varElement, _
36:                                                   intSDType, _
37:                                                   strIndent & "| ", _
38:                                                   boolDecipher
39:                 Next
40:              Else
41:
42:                 ' This is not an array, we deal with a Win32_Trustee
43:                 DisplayFormattedProperty Null, _
44:                                          strIndent & "| " & objWMIProperty.Name, _
45:                                          "(Win32_Trustee)", _
46:                                          Null
47:                 DecipherWMISecurityDescriptor objWMIProperty.Value, _
48:                                               intSDType, _
49:                                               strIndent & "| ", _
50:                                               boolDecipher
51:              End If
52:           Else
53:              Select Case Ucase (objWMIProperty.Name)
```

```
54:' Win32_SecurityDescriptor ----------------------------------------------------------------------
55:                     Case "CONTROLFLAGS"
56:                          If boolDecipher Then
57:                             DisplayFormattedProperty objWMIInstance, _
58:                                    strIndent & "| " & objWMIProperty.Name, _
59:                                    DecipherSDControlFlags (objWMIProperty.Value), _
60:                                    Null
61:                          Else
62:                             DisplayFormattedProperty objWMIInstance, _
63:                                    strIndent & "| " & objWMIProperty.Name, _
64:                                    "&h" & Hex (objWMIProperty.Value), _
65:                                    Null
66:                          End If
67:' Win32_ACE ---------------------------------------------------------------------------------------
68:                     Case "ACCESSMASK"
69:                          If boolDecipher Then
70:                             DisplayFormattedProperty objWMIInstance, _
71:                                    strIndent & "| " & objWMIProperty.Name, _
72:                                    DecipherACEMask (intSDType, objWMIProperty.Value), _
73:                                    Null
74:                          Else
75:                             DisplayFormattedProperty objWMIInstance, _
76:                                    strIndent & "| " & objWMIProperty.Name, _
77:                                    "&h" & Hex (objWMIProperty.Value), _
78:                                    Null
79:                          End If
80:                     Case "ACEFLAGS"
81:                          If boolDecipher Then
82:                             DisplayFormattedProperty objWMIInstance, _
83:                                    strIndent & "| " & objWMIProperty.Name, _
84:                                    DecipherACEFlags (intSDType, objWMIProperty.Value), _
85:                                    Null
86:                          Else
87:                             DisplayFormattedProperty objWMIInstance, _
88:                                    strIndent & "| " & objWMIProperty.Name, _
89:                                    "&h" & Hex (objWMIProperty.Value), _
90:                                    Null
91:                          End If
92:                     Case "ACETYPE"
93:                          If boolDecipher Then
94:                             DisplayFormattedProperty objWMIInstance, _
95:                                    strIndent & "| " & objWMIProperty.Name, _
96:                                    DecipherACEType (intSDType, objWMIProperty.Value), _
97:                                    Null
98:                          Else
99:                             DisplayFormattedProperty objWMIInstance, _
100:                                   strIndent & "| " & objWMIProperty.Name, _
101:                                   "&h" & Hex (objWMIProperty.Value), _
102:                                   Null
103:                         End If
104:' Win32_Trustee ---------------------------------------------------------------------------------
105:                    Case "SID"
106:                         DisplayFormattedProperty objWMIInstance, _
107:                               strIndent & "| " & objWMIProperty.Name, _
108:                               ConvertArrayInString (objWMIProperty.Value, ",", False), _
109:                               Null
110:' Default ---------------------------------------------------------------------------------------
111:                    Case Else
112:                         DisplayFormattedProperty objWMIInstance, _
113:                                strIndent & "| " & objWMIProperty.Name, _
```

```
114:                                    objWMIProperty.Name, _
115:                                    Null
116:                        End Select
117:            End If
118:        End If
119:    Next
120:
121:    WScript.Echo strIndent & "+-" & _
122:                 String (90 - Len (strIndent) + 2, "-")
...:
126:End Function
```

Sample 4.28 starts by enumerating all properties of the *Win32_SecurityDescriptor* (lines 24 through 119). Then the code checks for a value in the first property retrieved from the collection (line 25). If there is a value, the property type is examined, and, if the property contains an object instance (line 26), the script code verifies whether this property is an array (line 27). Since we know the WMI representation of a security descriptor, when the code discovers an array of objects, we know that we are dealing with one or more *Win32_ACE* instances, since it is the only property containing objects in an array (lines 29 through 39). If this is not an array (lines 42 through 50), then, based on our knowledge of the WMI security descriptor representation, we know that it's a *Win32_Trustee* instance. In both cases, because we deal with an instance, the DecipherWMISecurityDescriptor() function is recursively called (lines 35 through 38 or lines 47 through 50).

In case the property is not an object instance, the code displays its corresponding values (lines 53 through 116). Because the *Win32_SecurityDescriptor* instance with the various *Win32_Trustee* and *Win32_ACE* instances do not have properties using the same name, we can use the **Select Case** statement to convert and display the property accordingly. For instance, the *ControlFlags* property can be displayed without any bit flag interpretation (lines 62 through 65). If the **/Decipher+** switch is specified on the command line, its value can be deciphered (lines 57 through 60). The same logic applies for the *AccessMask*, the *AceFlags*, and the *AceType* properties (lines 68, 80, and 92). If the *SID* property must be displayed (line 105), since it contains a binary array, it is first converted to a comma-delimited string with the ConvertArrayInString() function, and then it is displayed (lines 112 through 115). Any other property is displayed by the default selection of the **Select Case** statement (lines 111 through 116).

As an end result, the next command line will display the security descriptor, as follows:

```
1:  C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
```

```
Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.

+- Win32_SecurityDescriptor ----------------------------------------------------------------
| ControlFlags: ..... &hB814
| DACL: ..... (Win32_ACE)
| +- Win32_ACE ------------------------------------------------------------------------------
| | AccessMask: ..... &h1F01FF
| | AceFlags: ..... &h3
| | AceType: ..... &h0
| | Trustee: ..... (Win32_Trustee)
| | +- Win32_Trustee ------------------------------------------------------------------------
| | | Domain: ..... BUILTIN
| | | Name: ..... Administrators
| | | SID: ..... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
| | | SidLength: ..... 16
| | | SIDString: ..... S-1-5-32-544
| | +-----------------------------------------------------------------------------------------
| +------------------------------------------------------------------------------------------
| +- Win32_ACE ------------------------------------------------------------------------------
| | AccessMask: ..... &h1200A9
| | AceFlags: ..... &h2
| | AceType: ..... &h0
| | Trustee: ..... (Win32_Trustee)
| | +- Win32_Trustee ------------------------------------------------------------------------
| | | Domain: ..... LISSWARENET
| | | Name: ..... MyGroup
| | | SID: ..... 1,5,0,0,0,...,246,207,122,236,255,136,223,4,0,0
| | | SidLength: ..... 28
| | | SIDString: ..... S-1-5-21-3533506287-3489020660-2298473594-1247
| | +-----------------------------------------------------------------------------------------
| +------------------------------------------------------------------------------------------
| Owner: ..... (Win32_Trustee)
| +- Win32_Trustee ------------------------------------------------------------------------
| | Domain: ..... BUILTIN
| | Name: ..... Administrators
| | SID: ..... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
| | SidLength: ..... 16
| | SIDString: ..... S-1-5-32-544
| +-----------------------------------------------------------------------------------------
| SACL: ..... (Win32_ACE)
| +- Win32_ACE ------------------------------------------------------------------------------
| | AccessMask: ..... &h10000
| | AceFlags: ..... &h43
| | AceType: ..... &h2
| | Trustee: ..... (Win32_Trustee)
| | +- Win32_Trustee ------------------------------------------------------------------------
| | | Domain: ..... BUILTIN
| | | Name: ..... Administrators
| | | SID: ..... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
| | | SidLength: ..... 16
| | | SIDString: ..... S-1-5-32-544
| | +-----------------------------------------------------------------------------------------
| +------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------
```

As we can see, the script also takes care of the WMI security descriptor display, since it encloses the various components between dashed lines to obtain a pseudographical representation.

4.10.2 Deciphering the ADSI security descriptor representation

To decipher an ADSI security descriptor representation, things are easier, since no recursive algorithm is used. Sample 4.29 implements the logic. Compared with Sample 4.28 ("Deciphering a WMI security descriptor representation"), the coding technique is more literal. For each ADSI COM object used to represent the security descriptor components, and for each of their properties, the code displays the corresponding value one by one. Essentially, Sample 4.29 makes use of the DisplayFormattedSTDProperty() function to format the output. This function has the exact same role as the DisplayFormattedProperty() function developed in Chapter 1 (Sample 1.6), but the DisplayFormattedSTDProperty() function is not related to a particular object model. It is a generic function to display information in the same way as the DisplayFormattedProperty() function (which is WMI related).

Sample 4.29 starts to display the properties of the **SecurityDescriptor** object (lines 23 through 33). Next, it continues with the **AccessControlEntry** object collection stored in the Discretionary ACL (lines 35 through 44). The code displays each property of the ACE in the DACL in a loop (lines 46 through 97). Then, it repeats the exact same logic applied to the ACE in the SACL (lines 109 through 172).

Sample 4.29 *Deciphering an ADSI security descriptor representation*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------
 9:Function DecipherADSISecurityDescriptor (objSD, intSDType, boolDecipher)
..:
19:    WScript.Echo strIndent & "+- ADSI Security Descriptor " & _
20:                 String (66 - Len (strIndent), "-")
21:
22:      Open Security Descriptor data ------------------------------------------------------
23:    DisplayFormattedSTDProperty  strIndent & "| Owner", objSD.Owner, Null
24:    DisplayFormattedSTDProperty  strIndent & "| Group", objSD.Group, Null
25:    DisplayFormattedSTDProperty  strIndent & "| Revision", objSD.Revision, Null
26:    If boolDecipher Then
27:       DisplayFormattedSTDProperty  strIndent & "| Control", _
28:                                    DecipherSDControlFlags (objSD.Control), _
29:                                    Null
30:    Else
31:       DisplayFormattedSTDProperty  strIndent & "| Control", _
32:                                    "&h" & Hex(objSD.Control), Null
33:    End If
34:
35:    intACECount = 0
```

```
36:    Set objACL = objSD.DiscretionaryAcl
37:    intACECount = objACL.AceCount
38:    If intACECount And Err.Number = 0 Then
39:       ' Open Discretionary ACL data ------------------------------------------------------------
40:       strIndent = strIndent & "|"
41:       WScript.Echo strIndent & "+- ADSI DiscretionaryAcl " & _
42:                    String (69 - Len (strIndent), "-")
43:
44:       strIndent = strIndent & "|"
45:
46:       For Each objACE In objACL
47:           ' Open ACE Data ---------------------------------------------------------------------
48:           WScript.Echo strIndent & "+- ADSI ACE " & _
49:                        String (82 - Len (strIndent), "-")
50:
51:           If boolDecipher Then
52:              DisplayFormattedSTDProperty  strIndent & "| AccessMask", _
53:                                           DecipherACEMask (intSDType, objACE.AccessMask), _
54:                                           Null
55:           Else
56:              DisplayFormattedSTDProperty  strIndent & "| AccessMask", _
57:                                           "&h" & Hex(objACE.AccessMask), Null
58:           End If
59:
60:           If boolDecipher Then
61:              DisplayFormattedSTDProperty  strIndent & "| AceFlags", _
62:                                           DecipherACEFlags (intSDType, objACE.AceFlags), _
63:                                           Null
64:           Else
65:              DisplayFormattedSTDProperty  strIndent & "| AceFlags", _
66:                                           "&h" & Hex(objACE.AceFlags), Null
67:           End If
68:
69:           If boolDecipher Then
70:              DisplayFormattedSTDProperty  strIndent & "| AceType", _
71:                                           DecipherACEType (intSDType, objACE.AceType), _
72:                                           Null
73:           Else
74:              DisplayFormattedSTDProperty  strIndent & "| AceType", _
75:                                           "&h" & Hex(objACE.AceType), Null
76:           End If
77:
78:           If boolDecipher Then
79:              DisplayFormattedSTDProperty  strIndent & "| AceFlagType", _
80:                                           DecipherACEFlagType (intSDType, objACE.Flags), _
81:                                           Null
82:           Else
83:              DisplayFormattedSTDProperty  strIndent & "| AceFlagType", _
84:                                           "&h" & Hex(objACE.Flags), Null
85:           End If
86:
87:           DisplayFormattedSTDProperty  strIndent & "| ObjectType", _
88:                                        objACE.ObjectType, Null
89:           DisplayFormattedSTDProperty  strIndent & "| InheritedObjectType", _
90:                                        objACE.InheritedObjectType, Null
91:           DisplayFormattedSTDProperty  strIndent & "| Trustee", _
92:                                        objACE.Trustee, Null
93:
94:           ' Close ACE Data --------------------------------------------------------------------
95:           WScript.Echo strIndent & "+-" & _
```

```
 96:                         String (90 - Len (strIndent) + 2, "-")
 97:       Next
 98:
 99:       strIndent = Mid (strIndent, 1, Len (strIndent) - 1)
100:
101:       ' Close Discretionary ACL data ------------------------------------------------------------
102:       WScript.Echo strIndent & "+-" & _
103:                    String (90 - Len (strIndent) + 2, "-")
104:       strIndent = Mid (strIndent, 1, Len (strIndent) - 1)
105:    Else
106:       Err.Clear
107:    End If
108:
109:    intACECount = 0
110:    Set objACL = objSD.SystemACL
111:    intACECount = objACL.AceCount
112:    If intACECount And Err.Number = 0 Then
113:       ' Open System ACL data --------------------------------------------------------------------
114:       strIndent = strIndent & "|"
115:       WScript.Echo strIndent & "+- ADSI SystemAcl " & _
116:                    String (76 - Len (strIndent), "-")
117:
118:       strIndent = strIndent & "|"
119:
120:       For Each objACE In objACL
...:
162:       Next
163:
164:       strIndent = Mid (strIndent, 1, Len (strIndent) - 1)
165:
166:       ' Close System ACL data -------------------------------------------------------------------
167:       WScript.Echo strIndent & "+-" & _
168:                    String (90 - Len (strIndent) + 2, "-")
169:       strIndent = Mid (strIndent, 1, Len (strIndent) - 1)
170:    Else
171:       Err.Clear
172:    End If
173:
174:    ' Close Security Descriptor data ------------------------------------------------------------
175:    WScript.Echo strIndent & "+-" & _
176:                 String (90 - Len (strIndent) + 2, "-")
177:
178:End Function
```

The end result with an ADSI representation is almost the same as the WMI representation. The difference resides in the SACL representation. The following sample output is a Folder security descriptor accessed under Windows 2000 with the **ADsSecurity.DLL**. As we have seen before, the **ADsSecurity.DLL** ActiveX component doesn't support the SACL access. Therefore, it is not displayed. The output obtained is as follows:

```
1:  C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /ADSI+
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Reading File or Folder security descriptor via ADSI from 'C:\MyDirectory'.
6:
```

```
 7:  +- ADSI Security Descriptor ----------------------------------------------------------------
 8:  | Owner: ..... BUILTIN\Administrators
 9:  | Group: ..... LISSWARENET\Domain Users
10:  | Revision: ..... 1
11:  | Control: ..... &h9004
12:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
13:  ||+- ADSI ACE ---------------------------------------------------------------------------
14:  ||| AccessMask: ..... &h1F01FF
15:  ||| AceFlags: ..... &h3
16:  ||| AceType: ..... &h0
17:  ||| AceFlagType: ..... &h0
18:  ||| Trustee: ..... BUILTIN\Administrators
19:  ||+---------------------------------------------------------------------------------------
20:  ||+- ADSI ACE ---------------------------------------------------------------------------
21:  ||| AccessMask: ..... &h1200A9
22:  ||| AceFlags: ..... &h2
23:  ||| AceType: ..... &h0
24:  ||| AceFlagType: ..... &h0
25:  ||| Trustee: ..... LISSWARENET\MyGroup
26:  ||+---------------------------------------------------------------------------------------
27:  |+-----------------------------------------------------------------------------------------
28:  +-----------------------------------------------------------------------------------------
```

4.11 Deciphering the security descriptor components

Retrieving the security descriptor components, such as the Access Control List and the Access Control Entries of the Discretionary ACL and the System ACL, is the very first step of the security descriptor deciphering. As shown in the two previous WMI and ADSI output representations, some properties contain numeric values. A closer look at these values shows that every bit composing the values has a specific meaning for the property. The interpretation of the properties represents the second step of the deciphering. In this section, we will decipher each value available from a security descriptor.

4.11.1 Deciphering the Owner and Group properties

The *Owner* and *Group* properties in the WMI object model are represented by a *Win32_Trustee* instance in an **SWBemObject** object. Therefore, Sample 4.28 ("Deciphering a WMI security descriptor representation"), by its recursive logic, naturally detects that the *Group* and *Owner* properties of the *Win32_SecurityDescriptor* class contain a *Win32_Trustee* instance. No particular deciphering technique is necessary. The *Win32_Trustee* instance is deciphered inside the DecipherWMISecurityDescriptor() in Sample 4.28. The following output sample shows the *Win32_Trustee* instances contained in a WMI *Owner* security descriptor representation coming from a folder (lines 69 through 76).

```
C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /Decipher+
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.

+- Win32_SecurityDescriptor ----------------------------------------------------------------
| ControlFlags: ..... &hB414
..:
| +------------------------------------------------------------------------------------------
| Owner: ..... (Win32_Trustee)
| +- Win32_Trustee ---------------------------------------------------------------------------
| | Domain: ..... BUILTIN
| | Name: ..... Administrators
| | SID: ..... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
| | SidLength: ..... 16
| | SIDString: ..... S-1-5-32-544
| +------------------------------------------------------------------------------------------
..:
..:
..:
```

When the security descriptor is represented in the ADSI object model, things are easier. The *Group* and the *Owner* properties of a security descriptor contain a literal string representing the trustee (i.e., Domain\User), which is displayed by the DecipherADSISecurityDescriptor() function in Sample 4.29 ("Deciphering an ADSI security descriptor representation"). If some SID resolution problems occur, the property could return a SID instead of a literal string representing the trustee. The following output sample shows the trustees contained in an ADSI security descriptor representation coming from a folder (line 8 for the trustee contained in the *Owner* property and line 9 for the trustee contained in the *Group* property).

```
C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /Decipher+ /ADSI+
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Reading File or Folder security descriptor via ADSI from 'C:\MyDirectory'.

+- ADSI Security Descriptor ----------------------------------------------------------------
| Owner: ..... BUILTIN\Administrators
| Group: ..... LISSWARENET\Domain Users
| Revision: ..... 1
| Control: ..... &h9404
..:
..:
..:
```

4.11.2 Deciphering the security descriptor Control Flags

The security descriptor *Control Flags* property (called *ControlFlags* with WMI and *Control* with ADSI; see Table 4.3, "The WMI and ADSI Secu-

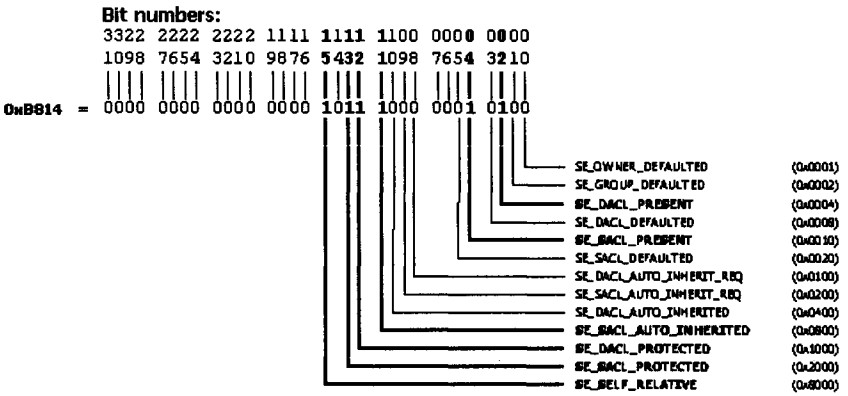
Table 4.8 *The Security Descriptor Control Flags Values*

SE_GROUP_DEFAULTED	0x2	A default mechanism, rather than the original provider of the security descriptor, provided the security descriptor's group SID. To set this flag, use the SetSecurityDescriptorGroup function.
SE_OWNER_DEFAULTED	0x1	A default mechanism, rather than the original provider of the security descriptor, provided the security descriptor's owner security identifier (SID). To set this flag, use the SetSecurityDescriptorOwner function.
SE_DACL_DEFAULTED	0x8	Indicates a security descriptor with a default DACL. For example, if an object's creator does not specify a DACL, the object receives the default DACL from the creator's access token. This flag can affect how the system treats the DACL, with respect to ACE inheritance. The system ignores this flag if the SE_DACL_PRESENT flag is not set. This flag is used to determine how the final DACL on the object is to be computed and is not stored physically in the security descriptor control of the securable object. To set this flag, use the SetSecurityDescriptorDacl function.
SE_SACL_DEFAULTED	0x20	A default mechanism, rather than the original provider of the security descriptor, provided the SACL. This flag can affect how the system treats the SACL, with respect to ACE inheritance. The system ignores this flag if the SE_SACL_PRESENT flag is not set. To set this flag, use the SetSecurityDescriptorSacl function.
SE_DACL_PRESENT	0x4	Indicates a security descriptor that has a DACL. If this flag is not set, or if this flag is set and the DACL is NULL, the security descriptor allows full access to everyone. This flag is used to hold the security information specified by a caller until the security descriptor is associated with a securable object. Once the security descriptor is associated with a securable object, the SE_DACL_PRESENT flag is always set in the security descriptor control. To set this flag, use SetSecurityDescriptorDacl.
SE_SACL_PRESENT	0x10	Indicates a security descriptor that has a SACL. To set this flag, use the SetSecurityDescriptorSacl function.
SE_DACL_PROTECTED	0x1000	Windows 2000/XP: Prevents the DACL of the security descriptor from being modified by inheritable ACEs. To set this flag, use the SetSecurityDescriptorControl function.
SE_SACL_PROTECTED	0x2000	Windows 2000/XP: Prevents the SACL of the security descriptor from being modified by inheritable ACEs. To set this flag, use the SetSecurityDescriptorControl function.
SE_DACL_AUTO_INHERIT_REQ	0x0100	Requests that the provider for the object protected by the security descriptor automatically propagate the DACL to existing child objects. If the provider supports automatic inheritance, it propagates the DACL to any existing child objects, and sets the SE_DACL_AUTO_INHERITED bit in the security descriptors of the object and its child objects.
SE_SACL_AUTO_INHERIT_REQ	0x0200	Requests that the provider for the object protected by the security descriptor automatically propagates the SACL to existing child objects. If the provider supports automatic inheritance, it propagates the SACL to any existing child objects, and sets the SE_SACL_AUTO_INHERITED bit in the security descriptors of the object and its child objects.
SE_DACL_AUTO_INHERITED	0x0400	Windows 2000/XP: Indicates a security descriptor in which the DACL is set up to support automatic propagation of inheritable ACEs to existing child objects. For Windows 2000 ACLs that support auto inheritance, this bit is always set. It is used to distinguish these ACLs from Windows NT 4.0 ACLs that do not support auto-inheritance. Protected servers can call the ConvertToAutoInheritPrivateObjectSecurity function to convert a security descriptor and set this flag. This bit is not set in security descriptors for Windows NT versions 4.0 and earlier, which do not support automatic propagation of inheritable ACEs.
SE_SACL_AUTO_INHERITED	0x0800	Windows 2000/XP: Indicates a security descriptor in which the SACL is set up to support automatic propagation of inheritable ACEs to existing child objects. The system sets this bit when it performs the automatic inheritance algorithm for the object and its existing child objects. Protected servers can call the ConvertToAutoInheritPrivateObjectSecurity function to convert a security descriptor and set this flag. This bit is not set in security descriptors for Windows NT versions 4.0 and earlier, which do not support automatic propagation of inheritable ACEs.
SE_SELF_RELATIVE	0x8000	Indicates a security descriptor in self-relative format with all the security information in a contiguous block of memory. If this flag is not set, the security descriptor is in absolute format. For more information, see Absolute and Self-Relative Security Descriptors.

rity Descriptor Exposed Methods and Properties") is helpful in determining the presence of the various security descriptor subcomponents, such as the DACL and SACL. With the introduction of Windows 2000, the security descriptor inheritance is also determined by this property. Each bit in the value has a specific meaning, summarized in Table 4.8.

Based on these values, the *Control Flags* bits must be deciphered with a bitwise operation, since each label in Table 4.8 corresponds to a specific bit setting in the value. For instance, Figure 4.18 shows flags that are turned ON or OFF when the *Control Flags* value equals 0xB814 (flags turned ON are in bold).

Figure 4.18 *The Control Flags bitwise values.*



Based on the flag values, Sample 4.30 deciphers the various bits of the property.

Sample 4.30 *Deciphering the security descriptor Control Flags property*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------
 9:Function DecipherSDControlFlags (intControlFlags)
..:
15:    strTemp = "&h" & Hex (intControlFlags)
16:
17:    If (intControlFlags And SE_OWNER_DEFAULTED) Then
18:       strTemp = strTemp & "," & "SE_OWNER_DEFAULTED"
19:    End If
20:    If (intControlFlags And SE_GROUP_DEFAULTED) Then
21:       strTemp = strTemp & "," & "SE_GROUP_DEFAULTED"
22:    End If
23:    If (intControlFlags And SE_DACL_PRESENT) Then
24:       strTemp = strTemp & "," & "SE_DACL_PRESENT"
25:    End If
26:    If (intControlFlags And SE_DACL_DEFAULTED) Then
27:       strTemp = strTemp & "," & "SE_DACL_DEFAULTED"
28:    End If
29:    If (intControlFlags And SE_SACL_PRESENT) Then
30:       strTemp = strTemp & "," & "SE_SACL_PRESENT"
31:    End If
32:    If (intControlFlags And SE_SACL_DEFAULTED) Then
33:       strTemp = strTemp & "," & "SE_SACL_DEFAULTED"
34:    End If
35:    If (intControlFlags And SE_DACL_AUTO_INHERIT_REQ) Then
36:       strTemp = strTemp & "," & "SE_DACL_AUTO_INHERIT_REQ"
37:    End If
38:    If (intControlFlags And SE_SACL_AUTO_INHERIT_REQ) Then
39:       strTemp = strTemp & "," & "SE_SACL_AUTO_INHERIT_REQ"
40:    End If
41:    If (intControlFlags And SE_DACL_AUTO_INHERITED) Then
42:       strTemp = strTemp & "," & "SE_DACL_AUTO_INHERITED"
43:    End If
```

```
44:    If (intControlFlags And SE_SACL_AUTO_INHERITED) Then
45:       strTemp = strTemp & "," & "SE_SACL_AUTO_INHERITED"
46:    End If
47:    If (intControlFlags And SE_DACL_PROTECTED) Then
48:       strTemp = strTemp & "," & "SE_DACL_PROTECTED"
49:    End If
50:    If (intControlFlags And SE_SACL_PROTECTED) Then
51:       strTemp = strTemp & "," & "SE_SACL_PROTECTED"
52:    End If
53:    If (intControlFlags And SE_SELF_RELATIVE) Then
54:       strTemp = strTemp & "," & "SE_SELF_RELATIVE"
55:    End If
56:
57:    DecipherSDControlFlags = ConvertStringInArray (strTemp, ",")
58:
59:End Function
```

Basically, the code performs a Boolean operation on the *Control Flags* value to determine the state of the bits corresponding to the flags listed in Table 4.8. If the bit is ON, the code constructs a comma-delimited string with the different bit labels from Table 4.8 (lines 15 through 55). Most values contained in a security descriptor or in one of its components use a similar deciphering technique. Only the flags used to decipher the value are different. Once completed, the comma-delimited string is converted to an array (line 57). The obtained output result would be as follows:

```
1:   C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /Decipher+
2:   Microsoft (R) Windows Script Host Version 5.6
3:   Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:   Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.
6:
7:   +- Win32_SecurityDescriptor ----------------------------------------------------------------
8:   | ControlFlags: ........................ &hB814
9:                                            SE_DACL_PRESENT
10:                                           SE_SACL_PRESENT
11:                                           SE_SACL_AUTO_INHERITED
12:                                           SE_DACL_PROTECTED
13:                                           SE_SACL_PROTECTED
14:                                           SE_SELF_RELATIVE
..:
..:
..:
```

The DecipherSDControlFlags() function is called by Sample 4.28 ("Deciphering a WMI security descriptor representation" at line 59) and Sample 4.29 ("Deciphering an ADSI security descriptor representation" at line 28). Note that both the WMI and ADSI deciphering functions call the DecipherSDControlFlags() function if the command-line parameter **/Decipher+** is specified. Independent of the object model representing the security descriptor, it makes sense to have the same meaning for the values in the properties.

The *Control Flags* property determines two behaviors of the security descriptor in regard to the inherited ACE:

- How the security descriptor behaves in regard to the ACE inherited from a parent object (i.e., Parent folder in the file system or Container in Active Directory). The SE_DACL_PROTECTED and SE_SACL_PROTECTED flags determine this first behavior.
- How the ACE defined in the security descriptor is inherited by the child objects (i.e., Subfolder in the file system or a child object in Active Directory). The SE_DACL_AUTO_INHERIT_REQ and the SE_SACL_AUTO_INHERIT_REQ flags determine this second behavior.

Since these flags are quite important for a security descriptor management, it could be useful to manage the SE_DACL_PROTECTED, SE_SACL_PROTECTED, the SE_DACL_AUTO_INHERIT_REQ, and the SE_SACL_AUTO_INHERIT_REQ flags. In order to configure these flags from the command line, it is necessary to calculate the new *Control Flags* value from the labels given on the command line. Sample 4.31 makes this calculation. Other *Control Flags* bits, listed in Table 4.8, are set by the script itself when necessary and are not configurable from the command line. This is why the array defined between lines 20 and 23 only contains the four flags just mentioned.

Sample 4.31 *Calculate the security descriptor controls value*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------
 9:Function CalculateSDControlFlags (arraySDControlFlags)
..:
20:    arraySDControlFlagsData = Array ("SE_DACL_PROTECTED", SE_DACL_PROTECTED, _
21:                                     "SE_SACL_PROTECTED", SE_SACL_PROTECTED, _
22:                                     "SE_DACL_AUTO_INHERIT_REQ", SE_DACL_AUTO_INHERIT_REQ, _
23:                                     "SE_SACL_AUTO_INHERIT_REQ", SE_SACL_AUTO_INHERIT_REQ)
24:
25:    For Each strSDControlFlags in arraySDControlFlags
26:        boolFlagFound = False
27:        For intIndice = 0 To UBound (arraySDControlFlagsData) Step 2
28:            If Ucase (strSDControlFlags) = Ucase (arraySDControlFlagsData(intIndice)) Then
29:               intSDControlFlags = intSDControlFlags + arraySDControlFlagsData(intIndice + 1)
30:               boolFlagFound = True
31:               Exit For
32:            End If
33:        Next
34:        If boolFlagFound = False Then
35:           WScript.Echo "Invalid SD control flags '" & strSDControlFlags & "'."
36:           WScript.Quit (1)
37:        End If
```

```
38:    Next
39:
40:    CalculateSDControlFlags = intSDControlFlags
41:
42:End Function
```

The flag labels given on the command line are passed in the form of an array as a parameter of the CalculateSDControlFlags() function (line 9). Next, another array is created (lines 20 through 23), which contains the flags accepted on the command line. To validate the flags given on the command line and calculate the final value, two loops are enclosed together (lines 25 through 38 and 27 through 33). If there is a match between the flag label given on the command line and the authorized list (line 28), its corresponding value is calculated (line 29). In case of invalid flag syntax, the loop detects that no match occurred and the script execution terminates (lines 34 through 37). For all flag labels given on the command line, the routine will follow the exact same logic. This logic is not related to the security descriptor *Control Flags* property. This algorithm is also used with the **/ACEType** and the **/ACEMask** switches in the CalculateACEType() CalculateACEMask() functions, respectively.

The CalculateSDControlFlags() function is only used when the *Control Flags* property is updated in the security descriptor. We will see later in section 4.12.3 ("Updating the security descriptor Control Flags") how this new value is updated in the security descriptor and how it is saved back to the secured entity.

Note that some security descriptors do not support ACE inheritance. This is, for instance, the case for every security descriptor of the Windows NT platform and for the file system share security descriptor (any platform). However, an ACE of a security descriptor from a parent CIM repository namespace is always inherited. This setting is not modifiable from the user interface.

4.11.3 Deciphering the Access Control Lists

Based on the security descriptor object model, the ACL is represented differently. A WMI security descriptor representation has a very basic representation of an ACL, since it is implemented in the form of an array exposed by the *DACL* and the *SACL* properties. Each array element contains a *Win32_ACE* instance. If an ACE must be added or removed from an ACL, the array must be manipulated accordingly. There is no WMI class explicitly representing an ACL.

An ADSI security descriptor representation is slightly different, since the *DiscretionaryACL* and *SystemACL* properties retrieve an **AccessControlList** object exposed by the **IADsAccessControlList** interface. This interface exposes ACEs as a collection. The interface also exposes methods to add and remove ACEs from the collection, which makes the ACE management in an ADSI ACL easier.

In both cases, there is no specific function to decipher an ACL. ACLs are retrieved in Sample 4.28 ("Deciphering a WMI security descriptor representation," lines 29 through 39) and Sample 4.29 ("Deciphering an ADSI security descriptor representation," lines 36 and 110).

For both object models, we will see the scripting technique to use to manage an ACE in an ACL in sections 4.12.4 ("Adding an ACE") and 4.12.5 ("Removing an ACE").

4.11.4 Deciphering the Access Control Entries

At the beginning of this chapter (see section 4.4.1, "The security descriptor WMI representation"), we saw that a security descriptor ACE is made up of six properties:

- The *ACE Trustee* property
- The *ACE Type* property
- The *ACE Flags* property
- The *ACE AccessMask* property
- The *ACE ObjectType* property
- The *ACE InheritedObjectType* property

However, the ADSI object model shows an additional property: the *ACE FlagType* property. The *ACE FlagType* property is used to determine the presence of a GUID number in the *ACE ObjectType* and *ACE InheritedObjectType* properties. This property is not a security descriptor component, but it is a property exposed by the ADSI security descriptor structural representation to signify the presence of a GUID number in *ObjectType* and/or *InheritedObjectType* ADSI properties.

As we can see, ACE properties are the same for any security descriptor regardless of its origin. On the other hand, property values and meanings may vary with the origin of the security descriptor (i.e., file system, registry, Active Directory). The best example is the *ACE AccessMask* property. The flags used to decipher an *ACE AccessMask* part of a file security descriptor

will be totally different from an *ACE AccessMask* part of an Active Directory security descriptor. In this section, we will discover how to decipher all ACE properties in relation to their origins. Some property deciphering techniques are common to all security descriptors (i.e., *ACE Trustee, ACE Type*); other property deciphering techniques will be unique to the origin of the security descriptor.

4.11.4.1 Deciphering the ACE Trustee property

As with the *Owner* and *Group* properties, the *ACE Trustee* property in the WMI object model is represented by a *Win32_Trustee* instance in an **SWBemObject** object. Therefore, Sample 4.28 ("Deciphering a WMI security descriptor representation"), by its recursive logic, naturally detects that the *Trustee* property of the *Win32_ACE* instance contains a *Win32_Trustee* instance. No particular bitwise deciphering technique is necessary. The *Win32_Trustee* instance is deciphered inside the DecipherWMISecurityDescriptor() in Sample 4.28.

When the security descriptor is represented in the ADSI object model, things are easier. The *Trustee* property contains a literal string representing the trustee (i.e., Domain\User). Therefore, Sample 4.29 ("Deciphering an ADSI security descriptor representation") makes a simple display of the string without further processing.

4.11.4.2 Deciphering the ACE Type property

The aim of the *ACE Type* property is to determine:

- If the ACE trustee part of the same ACE is granted for the rights specified in the *ACE AccessMask*.
- If the ACE trustee part of the same ACE is denied for the rights specified in the *ACE AccessMask*.
- If the ACE trustee part of the same ACE is audited for the rights specified in the *ACE AccessMask*.

Note that an ACE can only be used for one purpose at a time: granting, denying, or auditing. So, the *ACE Type* property does not use any bitwise operation for the deciphering, since only one of the three values can be assigned for one single ACE.

Sample 4.32 shows the DecipherACEType() function, which is called by Sample 4.28 ("Deciphering a WMI security descriptor representation") at line 96 and Sample 4.29 ("Deciphering an ADSI security

descriptor representation") at line 71. This demonstrates that the object model does not influence the interpretation of the value.

Sample 4.32 *Deciphering the ACE Type property*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function DecipherACEType (intSDType, intACEType)
..:
15:    strTemp = "&h" & Hex (intACEType)
16:
17:    Select Case intSDType
18:           Case cFileViaWMI, cFileViaADSI, _
19:                cShareViaWMI, _
20:                cShareViaADSI, _
21:                cRegistryViaADSI, _
22:                cWMINameSpaceViaWMI
23:                Select Case intACEType
24:                       Case ACCESS_ALLOWED_ACE_TYPE
25:                            strTemp = strTemp & "," & "ACCESS_ALLOWED_ACE_TYPE"
26:                       Case ACCESS_DENIED_ACE_TYPE
27:                            strTemp = strTemp & "," & "ACCESS_DENIED_ACE_TYPE"
28:                       Case SYSTEM_AUDIT_ACE_TYPE
29:                            strTemp = strTemp & "," & "SYSTEM_AUDIT_ACE_TYPE"
30:                       Case SYSTEM_ALARM_ACE_TYPE
31:                            strTemp = strTemp & "," & "SYSTEM_ALARM_ACE_TYPE"
32:                       Case Else
33:
34:                End Select
35:
36:           Case cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
37:                cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
38:                cExchange2000MailboxViaCDOEXM
39:                Select Case intACEType
40:                       Case ADS_ACETYPE_ACCESS_ALLOWED
41:                            strTemp = strTemp & "," & "ADS_ACETYPE_ACCESS_ALLOWED"
42:                       Case ADS_ACETYPE_ACCESS_DENIED
43:                            strTemp = strTemp & "," & "ADS_ACETYPE_ACCESS_DENIED"
44:                       Case ADS_ACETYPE_SYSTEM_AUDIT
45:                            strTemp = strTemp & "," & "ADS_ACETYPE_SYSTEM_AUDIT"
46:                       Case ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
47:                            strTemp = strTemp & "," & "ADS_ACETYPE_ACCESS_ALLOWED_OBJECT"
48:                       Case ADS_ACETYPE_ACCESS_DENIED_OBJECT
49:                            strTemp = strTemp & "," & "ADS_ACETYPE_ACCESS_DENIED_OBJECT"
50:                       Case ADS_ACETYPE_SYSTEM_AUDIT_OBJECT
51:                            strTemp = strTemp & "," & "ADS_ACETYPE_SYSTEM_AUDIT_OBJECT"
52:                       Case Else
53:
54:                End Select
55:
56:           Case cRegistryViaWMI, cWMINameSpaceViaADSI
57:
58:           Case Else
59:
```

```
60:    End Select
61:
62:    DecipherACEType = ConvertStringInArray (strTemp, ",")
63:
64:End Function
65:
..:
..:
..:
```

Sample 4.32 deciphers the *ACE Type* property according to the origin of the security descriptor. If the security descriptor does not originate from Active Directory, the code between lines 23 and 34 is executed. If the security descriptor is from Active Directory, the code between lines 39 and 54 is executed. As examples, for a non-Active Directory security descriptor, we will have an *ACE Type* (lines 18 through 34):

- Granting Access with the ACCESS_ALLOWED_ACE_TYPE flag.
- Denying Access with the ACCESS_DENIED_ACE_TYPE flag.
- Auditing Access with the SYSTEM_AUDIT_ACE_TYPE flag.

For instance, lines 38 and 54 show the *ACE Type* of a folder security descriptor.

```
 1:   C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /Decipher+ /ADSI+
 2:   Microsoft (R) Windows Script Host Version 5.6
 3:   Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:   Reading File or Folder security descriptor via ADSI from 'C:\MyDirectory'.
..:
16:   |+- ADSI DiscretionaryAcl ----------------------------------------------------------------
17:   ||+- ADSI ACE ---------------------------------------------------------------------------
18:   ||| AccessMask: ........................ &h1F01FF
..:
37:   ||| AceType: ............................ &h0
38:                                            ACCESS_ALLOWED_ACE_TYPE
39:   ||| AceFlagType: ....................... &h0
40:   ||| Trustee: ........................... BUILTIN\Administrators
41:   ||+---------------------------------------------------------------------------------------
42:   ||+- ADSI ACE ---------------------------------------------------------------------------
43:   ||| AccessMask: ........................ &h1200A9
..:
51:   ||| AceFlags: ........................... &h2
..:
53:   ||| AceType: ............................ &h0
54:                                            ACCESS_ALLOWED_ACE_TYPE
55:   ||| AceFlagType: ....................... &h0
56:   ||| Trustee: ........................... LISSWARENET\MyGroup
57:   ||+---------------------------------------------------------------------------------------
58:   |+-----------------------------------------------------------------------------------------
59:   +-----------------------------------------------------------------------------------------
```

If the security descriptor originates from Active Directory, we will have an *ACE Type* (lines 36 through 54):

- Granting Access with the ADS_ACETYPE_ACCESS_ALLOWED flag.
- Denying Access with the ADS_ACETYPE_ACCESS_ALLOWED flag.
- Auditing Access with the ADS_ACETYPE_SYSTEM_AUDIT flag.

For instance, lines 34 and 47 show the *ACE Type* of an Active Directory security descriptor.

```
 1:  C:\>WMIManageSD.Wsf /ADObject:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /Decipher+ /ADSI+
 2:  Microsoft (R) Windows Script Host Version 5.6
 3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:  Reading AD object security descriptor via ADSI from LDAP://CN=MyUser,CN=Users,...

 7:  +- ADSI Security Descriptor ----------------------------------------------------------------
 8:  | Owner: ............................... BUILTIN\Administrators
 9:  | Group: ............................... LISSWARENET\Alain.Lissoir
10:  | Revision: ............................ 1
11:  | Control: ............................. &h8C14
..:
17:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
18:  ||+- ADSI ACE ---------------------------------------------------------------------------
19:  ||| AccessMask: ........................ &hF01BD
..:
31:  ||| AceFlags: ......................... &h2
..:
33:  ||| AceType: .......................... &h0
34:                                         ADS_ACETYPE_ACCESS_ALLOWED
35:  ||| AceFlagType: ...................... &h0
36:  ||| Trustee: ......................... BUILTIN\Administrators
37:  ||+-------------------------------------------------------------------------------------
38:  ||+- ADSI ACE ---------------------------------------------------------------------------
39:  ||| AccessMask: ........................ &h20014
..:
43:  ||| AceFlags: ......................... &h3
..:
46:  ||| AceType: .......................... &h0
47:                                         ADS_ACETYPE_ACCESS_ALLOWED
48:  ||| AceFlagType: ...................... &h0
49:  ||| Trustee: ......................... LISSWARENET\MyUser
..:
..:
..:
```

If the security descriptor originates from the Active Directory and the ACE refers to Active Directory Extended Rights, we will have an *ACE Type* (lines 36 through 54):

- Granting Access with the ADS_ACETYPE_ACCESS_ALLOWED_OBJECT flag.

- Denying Access with the ADS_ACETYPE_ACCESS_ALLOWED_OBJECT flag.
- Auditing Access with the ADS_ACETYPE_SYSTEM_AUDIT_OBJECT flag.

For instance, lines 126 and 141 show the *ACE Type* of an Active Directory security descriptor for an Extended Right. Note the presence of a GUID for the *ACE ObjectType* property at lines 130 and 145.

```
  1:  C:\>WMIManageSD.Wsf /ADObject:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /Decipher+ /ADSI+
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:  Reading AD object security descriptor via ADSI from 'LDAP://CN=MyUser,CN=Users,...
  6:
  7:  +- ADSI Security Descriptor ----------------------------------------------------------------
  8:  | Owner: ............................... BUILTIN\Administrators
  9:  | Group: ............................... LISSWARENET\Alain.Lissoir
 10:  | Revision: ............................ 1
 11:  | Control: ............................. &h8C14
 ...:
 17:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
 18:  ||+- ADSI ACE ---------------------------------------------------------------------------
...:
119:  ||+- ADSI ACE ---------------------------------------------------------------------------
120:  ||| AccessMask: ........................ &h10
...:
122:  ||| AceFlags: ......................... &h12
...:
125:  ||| AceType: ........................... &h5
126:                                          ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
127:  ||| AceFlagType: ....................... &h3
...:
130:  ||| ObjectType: ......................... {037088F8-0AE1-11D2-B422-00A0C968F939}
131:  ||| InheritedObjectType: ................ {BF967ABA-0DE6-11D0-A285-00AA003049E2}
132:  ||| Trustee: ........................... BUILTIN\Pre-Windows 2000 Compatible Access
133:  ||+---------------------------------------------------------------------------------------
134:  ||+- ADSI ACE ---------------------------------------------------------------------------
135:  ||| AccessMask: ........................ &h10
...:
137:  ||| AceFlags: ......................... &h12
...:
140:  ||| AceType: ........................... &h5
141:                                          ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
142:  ||| AceFlagType: ....................... &h3
...:
145:  ||| ObjectType: ......................... {59BA2F42-79A2-11D0-9020-00C04FC2D3CF}
146:  ||| InheritedObjectType: ................ {BF967ABA-0DE6-11D0-A285-00AA003049E2}
147:  ||| Trustee: ........................... BUILTIN\Pre-Windows 2000 Compatible Access
148:  ||+---------------------------------------------------------------------------------------
...:
...:
...:
```

We will see in section 4.11.4.5.3 ("The Active Directory object ACE AccessMask property") how to manipulate the Active Directory *ACE AccessMask* property with Extended Rights.

4.11.4.3 Deciphering the ACE Flags property

The *ACE Flags* property determines the inheritance of an ACE. Do not confuse this property with the *Control Flags* property, which works at the security descriptor level, while the *ACE Flags* property works at the ACE level. The *ACE Flags* property determines how child objects inherit an ACE (i.e., Subfolder in the file system or a child object in Active Directory).

The DecipherACEFlags() function is called by Sample 4.28 ("Deciphering a WMI security descriptor representation") at line 84 and Sample 4.29 ("Deciphering an ADSI security descriptor representation") at line 62, which demonstrates once more that the object model does not influence the interpretation of the value.

As opposed to the *ACE Type* deciphering technique, the *ACE Flags* property is deciphered with a bitwise operation, because several bits determine how the ACE must be inherited. Even if the logic to decipher is always the same for any security descriptor, the origin determines the *ACE Flags* values to use to decipher. Table 4.9 lists the inheritance flags to use when the security has an origin other than Active Directory (i.e., files or folders, registry)

Table 4.10 lists the flags controlling ACE inheritance when the security descriptor comes from the Active Directory (i.e., Active Directory user object).

Moreover, the ACE inheritance capabilities rely on the security descriptor origin. For instance, a File System share security descriptor doesn't

Table 4.9 *The Security Descriptor Inheritance Flags*

OBJECT_INHERIT_ACE	0x1	Noncontainer objects contained by the primary object inherit the entry.
CONTAINER_INHERIT_ACE	0x2	Other containers that are contained by the primary object inherit the entry.
NO_PROPAGATE_INHERIT_ACE	0x4	The OBJECT_INHERIT_ACE and CONTAINER_INHERIT_ACE flags are not propagated to an inherited entry.
INHERIT_ONLY_ACE	0x8	The ACE does not apply to the primary object to which the ACL is attached, but objects contained by the primary object inherit the entry.
INHERITED_ACE	0x10	Only under Windows 2000, Windows XP, and Windows Server 2003, it indicates that the ACE was inherited. The system sets this bit when it propagates an inherited ACE to a child object.
SUCCESSFUL_ACCESS_ACE_FLAG	0x40	Used with system-audit ACEs in a SACL to generate audit messages for successful access attempts.
FAILED_ACCESS_ACE_FLAG	0x80	Used with system-audit ACEs in a SACL to generate audit messages for failed access attempts.
VALID_INHERIT_FLAGS	0x1F	Indicates whether the inherit flags are valid. The system sets this bit.

Table 4.10 *The Security Descriptor Inheritance Flags (Active Directory)*

ADS_ACEFLAG_INHERIT_ACE	0x2	Child objects will inherit this access-control entry (ACE). The inherited ACE is inheritable unless the ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE flag is set.
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	0x4	The system will clear the ADS_ACEFLAG_INHERIT_ACE flag for the inherited ACEs of child objects. This prevents the ACE from being inherited by subsequent generations of objects.
ADS_ACEFLAG_INHERIT_ONLY_ACE	0x8	Indicates an inherit-only ACE that does not exercise access control on the object to which it is attached. If this flag is not set, the ACE is an effective ACE that exerts access control on the object to which it is attached.
ADS_ACEFLAG_INHERITED_ACE	0x10	Indicates whether or not the ACE was inherited. The system sets this bit.
ADS_ACEFLAG_VALID_INHERIT_FLAGS	0x1F	Indicates whether the inherit flags are valid. The system sets this bit.
ADS_ACEFLAG_SUCCESSFUL_ACCESS	0x40	Generates audit messages for successful access attempts, used with ACEs that audit the system in a system access-control list (SACL).
ADS_ACEFLAG_FAILED_ACCESS	0x80	Generates audit messages for failed access attempts, used with ACEs that audit the system in a SACL.

implement the concept of inheritance, while an Active Directory security descriptor does. When we decipher the *ACE AccessMask* property, we will see how to set up the ACE inheritance, since it determines how *ACE AccessMask* is applied.

The DecipherACEFlags() function deciphering the *ACE Flags* is illustrated in Sample 4.33.

Sample 4.33 *Deciphering the ACE Flags property*

```
..:
..:
..:
65:
66:' ------------------------------------------------------------------------------------------
67:Function DecipherACEFlags (intSDType, intACEFlags)
..:
73:    strTemp = "&h" & Hex (intACEFlags)
74:
75:    Select Case intSDType
76:           Case cFileViaWMI, cFileViaADSI, _
77:                cShareViaWMI, _
78:                cShareViaADSI, _
79:                cRegistryViaADSI, _
80:                cWMINameSpaceViaWMI
81:                If (intACEFlags And OBJECT_INHERIT_ACE) Then
82:                   strTemp = strTemp & "," & "OBJECT_INHERIT_ACE"
83:                End If
84:                If (intACEFlags And CONTAINER_INHERIT_ACE) Then
85:                   strTemp = strTemp & "," & "CONTAINER_INHERIT_ACE"
86:                End If
87:                If (intACEFlags And NO_PROPAGATE_INHERIT_ACE) Then
88:                   strTemp = strTemp & "," & "NO_PROPAGATE_INHERIT_ACE"
89:                End If
..:
99:                If (intACEFlags And SUCCESSFUL_ACCESS_ACE_FLAG) Then
100:                  strTemp = strTemp & "," & "SUCCESSFUL_ACCESS_ACE_FLAG"
```

```
101:                End If
102:                If (intACEFlags And FAILED_ACCESS_ACE_FLAG) Then
103:                   strTemp = strTemp & "," & "FAILED_ACCESS_ACE_FLAG"
104:                End If
105:
106:          Case cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
107:                cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
108:                cExchange2000MailboxViaCDOEXM
109:                If (intACEFlags And ADS_ACEFLAG_OBJECT_INHERIT_ACE) Then
110:                   strTemp = strTemp & "," & "ADS_ACEFLAG_OBJECT_INHERIT_ACE"
111:                End If
112:                If (intACEFlags And ADS_ACEFLAG_CONTAINER_INHERIT_ACE) Then
113:                   strTemp = strTemp & "," & "ADS_ACEFLAG_CONTAINER_INHERIT_ACE"
114:                End If
115:                If (intACEFlags And ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE) Then
116:                   strTemp = strTemp & "," & "ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE"
117:                End If
...:
127:                If (intACEFlags And ADS_ACEFLAG_SUCCESSFUL_ACCESS) Then
128:                   strTemp = strTemp & "," & "ADS_ACEFLAG_SUCCESSFUL_ACCESS"
129:                End If
130:                If (intACEFlags And ADS_ACEFLAG_FAILED_ACCESS) Then
131:                   strTemp = strTemp & "," & "ADS_ACEFLAG_FAILED_ACCESS"
132:                End If
133:
134:          Case cRegistryViaWMI, cWMINameSpaceViaADSI
135:
136:          Case Else
137:
138:    End Select
139:
140:    DecipherACEFlags = ConvertStringInArray (strTemp, ",")
141:
142:End Function
143:
...:
...:
...:
```

The following output shows the *ACE Flags* values from lines 34 through 36 and at lines 51 and 52.

```
  1:   C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /Decipher+ /ADSI+
  2:   Microsoft (R) Windows Script Host Version 5.6
  3:   Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:   Reading File or Folder security descriptor via ADSI from 'C:\MyDirectory'.
  6:
  7:   +- ADSI Security Descriptor ----------------------------------------------------------------
  8:   | Owner: ............................. BUILTIN\Administrators
  9:   | Group: ............................. LISSWARENET\Domain Users
 10:   | Revision: .......................... 1
 11:   | Control: ........................... &h9404
 ..:
 16:   |+- ADSI DiscretionaryAcl ---------------------------------------------------------------
 17:   ||+- ADSI ACE -------------------------------------------------------------------------
 18:   ||| AccessMask: ....................... &h1F01FF
 ..:
 34:   ||| AceFlags: .......................... &h3
```

```
35:                                          OBJECT_INHERIT_ACE
36:                                          CONTAINER_INHERIT_ACE
37:   ||| AceType: ........................... &h0
..:
39:   ||| AceFlagType: ....................... &h0
40:   ||| Trustee: ........................... BUILTIN\Administrators
41:   ||+-------------------------------------------------------------------------------------
42:   ||+- ADSI ACE ---------------------------------------------------------------------------
43:   ||| AccessMask: ........................ &h1200A9
..:
51:   ||| AceFlags: .......................... &h2
52:                                          CONTAINER_INHERIT_ACE
53:   ||| AceType: ........................... &h0
..:
55:   ||| AceFlagType: ....................... &h0
56:   ||| Trustee: ........................... LISSWARENET\MyGroup
57:   ||+-------------------------------------------------------------------------------------
58:   |+--------------------------------------------------------------------------------------
59:   +---------------------------------------------------------------------------------------
```

4.11.4.4 Deciphering the ACE FlagType property

The *ACE FlagType* is only used when the *ACE ObjectType* or *ACE InheritedObjectType* properties contain a GUID number. Only Sample 4.29 ("Deciphering an ADSI security descriptor representation") at line 80 calls the DecipherACEFlagType() function. The security descriptor WMI representation supports the display of a GUID number but does not use an *ACE FlagType* property. This property is a peculiarity of the ADSI object model representation. We will see in section 4.11.4.5.3 ("The Active Directory object ACE AccessMask property") how to interpret the GUID number. Except for these peculiarities, the *ACE FlagType* coding and deciphering technique are always the same (see Sample 4.34).

Sample 4.34 *Deciphering the ACE FlagType property*

```
...:
...:
...:
143:
144:' ---------------------------------------------------------------------------------------
145:Function DecipherACEFlagType (intSDType, intACEFlagType)
...:
151:    strTemp = "&h" & Hex (intACEFlagType)
152:
153:    Select Case intSDType
154:          Case cFileViaWMI, cFileViaADSI, _
155:               cShareViaWMI, _
156:               cShareViaADSI, _
157:               cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
158:               cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
159:               cExchange2000MailboxViaCDOEXM, _
160:               cRegistryViaADSI, _
161:               cWMINameSpaceViaWMI
162:               If (intACEFlagType And ADS_FLAG_OBJECT_TYPE_PRESENT) Then
```

```
163:                   strTemp = strTemp & "," & "ADS_FLAG_OBJECT_TYPE_PRESENT"
164:                End If
165:                If (intACEFlagType And ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT) Then
166:                   strTemp = strTemp & "," & "ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT"
167:                End If
168:
169:          Case cRegistryViaWMI, cWMINameSpaceViaADSI
170:
171:          Case Else
172:
173:    End Select
174:
175:    DecipherACEFlagType = ConvertStringInArray (strTemp, ",")
176:
177:End Function
178:
...:
...:
...:
```

The following output sample shows the *ACE FlagType* values from line 157 through 159 and 172 through 174.

```
  1:   C:\>WMIManageSD.Wsf /ADObject:"CN=MyUser,CN=Users,DC=LissWare,DC=Net" /Decipher+ /ADSI+
  2:   Microsoft (R) Windows Script Host Version 5.6
  3:   Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:   Reading AD object security descriptor via ADSI from 'LDAP://CN=MyUser,CN=Users,...
  6:
  7:   +- ADSI Security Descriptor ----------------------------------------------------------------
  8:   | Owner: ............................... BUILTIN\Administrators
  9:   | Group: ............................... LISSWARENET\Alain.Lissoir
 10:   | Revision: ............................ 1
 11:   | Control: ............................. &h8C14
 ..:
 17:   |+- ADSI DiscretionaryAcl ---------------------------------------------------------------
149:   ||+- ADSI ACE ------------------------------------------------------------------------------
150:   ||| AccessMask: ........................ &h10
...:
152:   ||| AceFlags: .......................... &h12
...:
155:   ||| AceType: ........................... &h5
...:
157:   ||| AceFlagType: ....................... &h3
158:                                            ADS_FLAG_OBJECT_TYPE_PRESENT
159:                                            ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
160:   ||| ObjectType: ........................ {BC0AC240-79A9-11D0-9020-00C04FC2D4CF}
161:   ||| InheritedObjectType: ............... {BF967ABA-0DE6-11D0-A285-00AA003049E2}
162:   ||| Trustee: .......................... BUILTIN\Pre-Windows 2000 Compatible Access
163:   ||+------------------------------------------------------------------------------------------
164:   ||+- ADSI ACE ------------------------------------------------------------------------------
165:   ||| AccessMask: ........................ &h10
...:
167:   ||| AceFlags: .......................... &h12
...:
170:   ||| AceType: ........................... &h5
...:
172:   ||| AceFlagType: ....................... &h3
173:                                            ADS_FLAG_OBJECT_TYPE_PRESENT
```

```
174:                                          ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
175:   ||| ObjectType: ........................ {4C164200-20C0-11D0-A768-00AA006E0529}
176:   ||| InheritedObjectType: ............... {BF967ABA-0DE6-11D0-A285-00AA003049E2}
177:   ||| Trustee: ........................... BUILTIN\Pre-Windows 2000 Compatible Access
178:   ||+-------------------------------------------------------------------------------
...:
...:
...:
```

4.11.4.5 Deciphering the ACE AccessMask property

To decipher security descriptors, the script makes use of the DecipherACEMask() function. The DecipherACEMask() is divided into several samples due to the fact that there is always a specific set of flags to use for each security descriptor origin. The DecipherACEMask() function is illustrated in Samples 4.35 through 4.40 in the following sections.

This DecipherACEMask() function is called by Sample 4.28 ("Deciphering a WMI security descriptor representation") at line 72 and Sample 4.29 ("Deciphering an ADSI security descriptor representation") at line 53.

4.11.4.5.1 The files and folders ACE AccessMask property

To decipher a file or a folder *ACE AccessMask*, it is necessary to use the flag values listed in Table 4.11.

The column headings in this table represent the settings in the user interface. The left column lists the flags that must be used to decipher or set an *ACE AccessMask* value for a file or a folder, while the top row shows the user interface selection.

In Figure 4.19, the user interface shows a folder security descriptor. We see that the "Read & Execute" right is granted to "Everyone." In Table 4.11, in the column "Read & Execute," we see that each time there is a cross in the cell the corresponding flag is set. In such a case, we have the following flags:

- FOLDER_LIST_DIRECTORY
- FILE_READ_EA
- FOLDER_TRAVERSE
- FILE_READ_ATTRIBUTES
- FILE_READ_CONTROL
- FILE_SYNCHRONIZE

To ease flag use, some flags are generic. They are made from a combination of several flags. So, instead of using all previously listed flags, the "Read

Table 4.11 *The Files and Folders ACE AccessMask Values*

Granted & denied rights		Standard View						Advanced View												
		Full Control	Modify	Read & Execute	List Folder Contents	Read	Write	Traverse Folder / Execute File	List Folder / Execute Data	Read Attributes	Read Extended Attributes	Create Files / Write Data	Create Folders / Append Data	Write Attributes	Write Extended Attributes	Delete Subfolders and Files	Delete	Read Permissions	Change Permissions	Take Ownership
ACEType																				
ACCESS_ALLOWED_ACE_TYPE (Allowed access ACE)	0x0																			
ACCESS_DENIED_ACE_TYPE (Denied access ACE)	0x1	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]
SYSTEM_AUDIT_ACE_TYPE (System Audit ACE)	0x2																			
ACEMask																				
FILE_GENERIC_EXECUTE	0x1200A9			X																
FILE_GENERIC_READ	0x120089					X														
FILE_GENERIC_WRITE	0x100116						X													
FILE_ALL_ACCESS	0x1F01FF	X																		
FILE_APPEND_DATA (FOLDER_ADD_SUBDIRECTORY)	0x000004	X	X				X						X							
FILE_DELETE	0x010000	X	X														X			
FILE_DELETE_CHILD	0x000040	X														X				
FILE_EXECUTE (FOLDER_TRAVERSE)	0x000020	X	X	X	X			X												
FILE_READ_ATTRIBUTES	0x000080	X	X	X	X	X				X										
FILE_READ_CONTROL	0x020000	X	X	X	X	X												X		
FILE_READ_DATA (FOLDER_LIST_DIRECTORY)	0x000001	X	X	X	X	X			X											
FILE_READ_EA	0x000008	X	X	X	X	X					X									
FILE_SYNCHRONIZE	0x100000	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FILE_WRITE_ATTRIBUTES	0x000100	X	X				X							X						
FILE_WRITE_DAC	0x040000	X																	X	
FILE_WRITE_DATA (FOLDER_ADD_FILE)	0x000002	X	X				X						X							
FILE_WRITE_EA	0x000010	X	X				X								X					
FILE_WRITE_OWNER	0x080000	X																		X

(1) **Windows NT 4.0/Windows 2000:** The **ADsSecurity.DLL** from the ADSI Resource Kit does not retrieve the SACL object from the registry.
Windows XP/Windows Server 2003: Unfortunately, a bug in the ADsSecurityUtility interface prevents the retrieval of the SystemACL. Microsoft doesn't plan to fix this bug in the RTM code for timing issues. WMI offers an acceptable work-around for file and folders only. For the registry key, there is no work-around available unless you use the UserRight.Control developed to work around this problem. (See section 4.7.1.2, "Retrieving file and folder security descriptors with ADSI.")

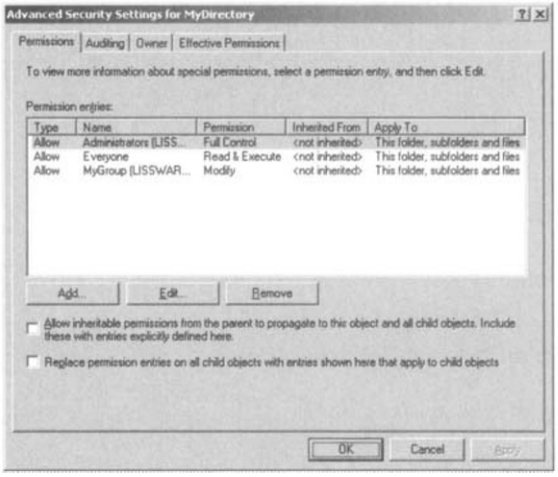


Figure 4.19 *The files and folders security descriptor user interface.*

& Execute" right can be summarized by the use of the FILE_GENERIC_EXECUTE flag. For the files and folders, there are three generic flags listed in Table 4.11 (FILE_GENERIC_EXECUTE, FILE_GENERIC_READ, and FILE_GENERIC_WRITE).

To decipher the *ACE AccessMask* property, the logic is always the same. Sample 4.35 shows the bitwise operations executed with the flags listed in Table 4.11. Again, we see that the deciphering technique applies for an *ACE AccessMask* coming from a WMI security descriptor or an ADSI security descriptor representation.

Sample 4.35 *Deciphering the ACE AccessMask property for files and folders*

```
...:
...:
...:
178:
179:' ------------------------------------------------------------------------------------------
180:Function DecipherACEMask (intSDType, intACEMask)
...:
186:    strTemp = "&h" & Hex (intACEMask)
187:
188:    Select Case intSDType
189:           Case cFileViaWMI, cFileViaADSI
190:                If (intACEMask = FILE_ALL_ACCESS) Then
191:                   strTemp = strTemp & "," & "(FILE_ALL_ACCESS)"
192:                End If
193:                If (intACEMask = FILE_GENERIC_EXECUTE) Then
194:                   strTemp = strTemp & "," & "(FILE_GENERIC_EXECUTE)"
195:                End If
196:                If (intACEMask = FILE_GENERIC_READ) Then
197:                   strTemp = strTemp & "," & "(FILE_GENERIC_READ)"
198:                End If
199:                If (intACEMask = FILE_GENERIC_WRITE) Then
200:                   strTemp = strTemp & "," & "(FILE_GENERIC_WRITE)"
201:                End If
202:                If (intACEMask And FILE_READ_DATA) Then
203:                   strTemp = strTemp & "," & "FILE_READ_DATA " & _
204:                                             "(FOLDER_LIST_DIRECTORY for a Folder)"
205:                End If
206:                If (intACEMask And FILE_WRITE_DATA) Then
207:                   strTemp = strTemp & "," & "FILE_WRITE_DATA " & _
208:                                             "(FOLDER_ADD_FILE for a Folder)"
209:                End If
210:                If (intACEMask And FILE_APPEND_DATA) Then
211:                   strTemp = strTemp & "," & "FILE_APPEND_DATA " & _
212:                                             "(FOLDER_ADD_SUBDIRECTORY for a Folder)"
213:                End If
...:
245:                If (intACEMask And FILE_SYNCHRONIZE) Then
246:                   strTemp = strTemp & "," & "FILE_SYNCHRONIZE"
247:                End If
248:
...:
...:
...:
```

To distinguish the security descriptor origin, the DecipherACEMask() function uses a **Select Case** statement, where each case corresponds to a deciphering of an *ACE AccessMask* value from a specific security descriptor origin.

The execution of the following command line will completely decipher the security descriptor shown in Figure 4.19.

```
C:\>WMIManageSD.wsf /FileSystem:C:\MyDirectory /Decipher+
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.

+- Win32_SecurityDescriptor ----------------------------------------------------------------
| ControlFlags: ..................... &hBC14
                                      SE_DACL_PRESENT
                                      SE_SACL_PRESENT
                                      SE_DACL_AUTO_INHERITED
                                      SE_SACL_AUTO_INHERITED
                                      SE_DACL_PROTECTED
                                      SE_SACL_PROTECTED
                                      SE_SELF_RELATIVE
| DACL: ............................. (Win32_ACE)
| +- Win32_ACE ------------------------------------------------------------------------------
| | AccessMask: ..................... &h1F01FF
                                      (FILE_ALL_ACCESS)
                                      FOLDER_LIST_DIRECTORY
                                      FOLDER_ADD_FILE
                                      FOLDER_ADD_SUBDIRECTORY
                                      FILE_READ_EA
                                      FILE_WRITE_EA
                                      FOLDER_TRAVERSE
                                      FILE_DELETE_CHILD
                                      FILE_READ_ATTRIBUTES
                                      FILE_WRITE_ATTRIBUTES
                                      FILE_DELETE
                                      FILE_READ_CONTROL
                                      FILE_WRITE_DAC
                                      FILE_WRITE_OWNER
                                      FILE_SYNCHRONIZE
| | AceFlags: ....................... &h3
                                      OBJECT_INHERIT_ACE
                                      CONTAINER_INHERIT_ACE
| | AceType: ........................ &h0
                                      ACCESS_ALLOWED_ACE_TYPE
| | Trustee: ........................ (Win32_Trustee)
| | +- Win32_Trustee --------------------------------------------------------------------------
| | | Domain: ....................... BUILTIN
| | | Name: ......................... Administrators
| | | SID: .......................... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
| | | SidLength: .................... 16
| | | SIDString: .................... S-1-5-32-544
| | +-------------------------------------------------------------------------------------------
| +-----------------------------------------------------------------------------------------
| +- Win32_ACE ------------------------------------------------------------------------------
| | AccessMask: ..................... &h1200A9
                                      (FILE_GENERIC_EXECUTE)
```

```
                                          FOLDER_LIST_DIRECTORY
                                          FILE_READ_EA
                                          FOLDER_TRAVERSE
                                          FILE_READ_ATTRIBUTES
                                          FILE_READ_CONTROL
                                          FILE_SYNCHRONIZE
  | | AceFlags: ........................... &h3
                                          OBJECT_INHERIT_ACE
                                          CONTAINER_INHERIT_ACE
  | | AceType: ............................ &h0
                                          ACCESS_ALLOWED_ACE_TYPE
  | | Trustee: ............................ (Win32_Trustee)
  | | +- Win32_Trustee ------------------------------------------------------------------------
  | | | Name: ............................. Everyone
  | | | SID: .............................. 1,1,0,0,0,0,0,1,0,0,0,0
  | | | SidLength: ........................ 12
  | | | SIDString: ........................ S-1-1-0
  | | +-----------------------------------------------------------------------------------------
  | +-------------------------------------------------------------------------------------------
  | +- Win32_ACE ----------------------------------------------------------------------------
  | | AccessMask: ......................... &h1301BF
                                          FOLDER_LIST_DIRECTORY
                                          FOLDER_ADD_FILE
                                          FOLDER_ADD_SUBDIRECTORY
                                          FILE_READ_EA
                                          FILE_WRITE_EA
                                          FOLDER_TRAVERSE
                                          FILE_READ_ATTRIBUTES
                                          FILE_WRITE_ATTRIBUTES
                                          FILE_DELETE
                                          FILE_READ_CONTROL
                                          FILE_SYNCHRONIZE
  | | AceFlags: ........................... &h3
                                          OBJECT_INHERIT_ACE
                                          CONTAINER_INHERIT_ACE
  | | AceType: ............................ &h0
                                          ACCESS_ALLOWED_ACE_TYPE
  | | Trustee: ............................ (Win32_Trustee)
  | | +- Win32_Trustee ------------------------------------------------------------------------
  | | | Domain: ........................... LISSWARENET
  | | | Name: ............................. MyGroup
  | | | SID: .............................. 1,5,0,0,0,...,207,122,236,255,136,223,4,0,0
  | | | SidLength: ........................ 28
  | | | SIDString: ........................ S-1-5-21-3533506287-3489020660-2298473594-1247
  | | +-----------------------------------------------------------------------------------------
  | +-------------------------------------------------------------------------------------------
  | Owner: .............................. (Win32_Trustee)
  | +- Win32_Trustee ------------------------------------------------------------------------
  | | Domain: ........................... BUILTIN
  | | Name: ............................. Administrators
  | | SID: .............................. 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
  | | SidLength: ........................ 16
  | | SIDString: ........................ S-1-5-32-544
  | +-----------------------------------------------------------------------------------------
  +---------------------------------------------------------------------------------------------
```

As we have seen, the *ACE AccessMask* inheritance is defined by the *ACE Flags* property. Although the deciphering technique is the same for any security descriptor (see Sample 4.33, "Deciphering the *ACE Flags*

Table 4.12 *The Files and Folders ACE Flags Values*

Inheritance & Audit		(Folders only)								
		This folder only	This folder, subfolders, and files	This folder and subfolders	This folder and files	Subfolders and files only	Subfolders only	Files only	Audit Successful access	Audit Failed access
ACEFlags										
NONE	0x0	X								
CONTAINER_INHERIT_ACE	0x2		X	X		X	X			
INHERIT_ONLY_ACE	0x8					X	X	X		
INHERITED_ACE[1]	0x10									
NO_PROPAGATE_INHERIT_ACE	0x4									
OBJECT_INHERIT_ACE	0x1		X		X	X		X		
VALID_INHERIT_FLAG[1]	0x1F									
SUCCESSFUL_ACCESS_ACE_FLAG	0x40								X	
FAILED_ACCESS_ACE_FLAG	0x80									X

(1) Set by the system.

property"), the flag values used and their combinations to decipher or set the *ACE Flags* property are dependent on the security descriptor origin, since the origin determines the inheritance capabilities.

Table 4.12 summarizes the flag values in regard to the inheritance settings that can be set from the user interface shown in Figure 4.20. The previous security descriptor deciphering output of Figure 4.19 shows the *ACE Flags* settings for the configured inheritance.

To set up a security descriptor in a folder similar to the one shown in Figure 4.19, the script must be executed several times, since it sets only one ACE at a time. Of course, Tables 4.11 ("The files and folders *ACE AccessMask* values") and 4.12 ("The files and folders *ACE Flags* values") can be used to determine these settings. In such a case, the following command lines will set up the settings in Figure 4.19:

```
1:  WMIManageSD.wsf /FileSystem:C:\MyDirectory /Trustee:REMOVE_ALL_ACE /DelAce+
2:
3:  WMIManageSD.wsf /FileSystem:C:\MyDirectory /Trustee:BUILTIN\Administrators
4:                  /ACEType:ACCESS_ALLOWED_ACE_TYPE
5:                  /ACEMask:FILE_ALL_ACCESS
6:                  /ACEFlags:OBJECT_INHERIT_ACE,CONTAINER_INHERIT_ACE
7:                  /AddAce+
8:
9:  WMIManageSD.wsf /FileSystem:C:\MyDirectory /Trustee:Everyone /DelAce+
10:
11: WMIManageSD.wsf /FileSystem:C:\MyDirectory /Trustee:LissWareNET\Everyone
12:                 /ACEType:ACCESS_ALLOWED_ACE_TYPE
13:                 /ACEMask:FILE_GENERIC_EXECUTE
14:                 /ACEFlags:OBJECT_INHERIT_ACE,CONTAINER_INHERIT_ACE
```

```
15:                 /AddAce+
16:
17:   WMIManageSD.wsf /FileSystem:C:\MyDirectory /Trustee:LissWareNET\MyGroup
18:                 /ACEType:ACCESS_ALLOWED_ACE_TYPE
19:                 /ACEMask:FOLDER_LIST_DIRECTORY,
20:                         FOLDER_ADD_FILE,FOLDER_ADD_SUBDIRECTORY,FILE_READ_EA,FILE_WRITE_EA,
21:                         FOLDER_TRAVERSE,FILE_READ_ATTRIBUTES,FILE_WRITE_ATTRIBUTES,
22:                         FILE_DELETE,FILE_READ_CONTROL,FILE_SYNCHRONIZE
23:                 /ACEFlags:OBJECT_INHERIT_ACE,CONTAINER_INHERIT_ACE /AddAce+
```

Please take a few minutes to compare the command-line settings with the previous output and the content of Tables 4.11 and 4.12.

At line 1, the script removes all available ACEs. We will see in section 4.12.5 ("Removing an ACE") that the end result of this operation sets a full access right to "Everyone" on the secured object. Although it is possible to remove all ACEs one by one to obtain the desired configuration, this makes the work more complicated, because it forces us to know which ACE has to be removed. By removing all ACE entries at once, we start the security

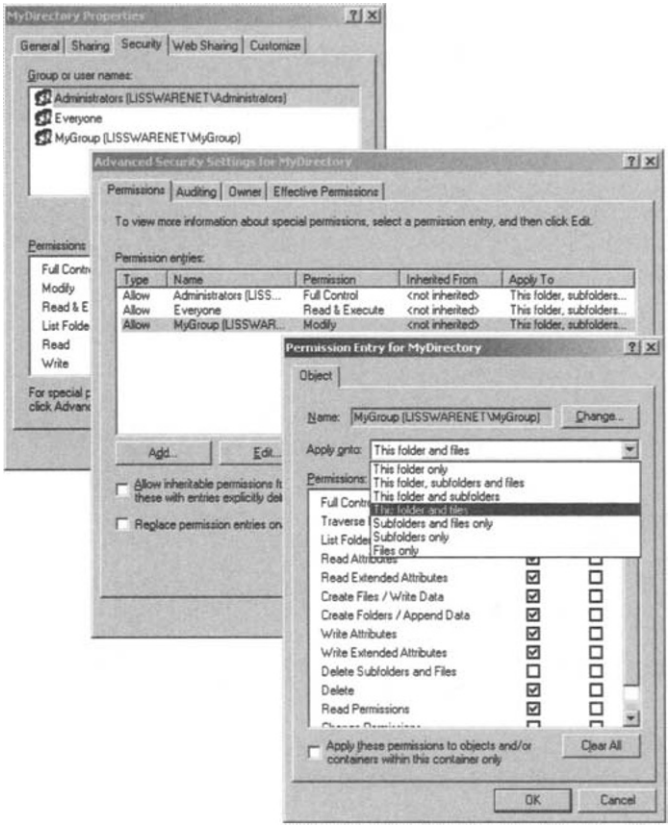


Figure 4.20 *The files and folders inheritance user interface.*

descriptor configuration from a clear and known situation. From line 3 through 7, the script configures the "Administrators" group with a full access right. If the user configuring the security descriptor is part of the "Administrators" group (which is supposed to be in this example), then the "Everyone" group can be removed (line 9). Of course, as shown in Figure 4.19, the "Everyone" group has a "Read & Execute" access. This configuration is set up from line 11 through 15. Although it is technically possible to edit the ACE properties of the deleted ACE at line 9, this requires more granularity in terms of management capabilities to change existing ACE-specific properties. For the sake of simplicity, the script manages the ACE security descriptor at the ACE level for an existing ACE, not at the ACE property level. Finally, from line 17 through 23, the script grants the "Modify" right to the "MyGroup" group. Once completed, we obtain the security settings shown in Figure 4.19. We will see in section 4.13 ("Updating the security descriptor") how the security descriptor is saved back to the secured entity (which is a folder in this example).

The script accesses the security descriptor via WMI and therefore uses the WMI *Security* provider and its related class methods implementing the security descriptor access. In the case of a file or a folder, it is possible to use ADSI as the access method. Therefore, the switch **/ADSI+** must be specified. Keep in mind the restrictions that apply to the SACL access when using ADSI (see Table 4.4).

Now, if you compare the results obtained when deciphering the Figure 4.19 configuration with the command-line switches used previously, you will see that the exact same settings are used. Basically, each time you need to configure a security descriptor, it is a good idea to configure the desired result via the user interface first. Next, run the script to decipher the desired result and reuse this output to customize the command-line switches to automate the security configuration settings. We will see other examples with other security descriptors later. The logic is always the same. Only some flags related to a specific security descriptor must be adapted (file, share, Active Directory objects, etc.).

4.11.4.5.2 The File System share ACE AccessMask property

To decipher a File System share *ACE AccessMask*, it is necessary to use the flag values listed in Table 4.13.

Deciphering the File System share *ACE AccessMask* is quite easy, since there are only three flags used. Sample 4.36 is the continuation of Sample 4.35 ("Deciphering the *ACE AccessMask* property for files and folders") and

Table 4.13 *The File System Share ACE AccessMask Values*

Granted & denied rights		Standard View		
		Full Control	Change	Read
ACEType				
ACCESS_ALLOWED_ACE_TYPE	0x0	X	X	X
ACCESS_DENIED_ACE_TYPE	0x1			
ACEMask				
FILE_SHARE_FULL_ACCESS	0x0C0040	X		
FILE_SHARE_CHANGE_ACCESS	0x010116	X	X	
FILE_SHARE_READ_ACCESS	0x1200A9	X	X	X

shows how to decipher a File System share *ACE AccessMask* with the values listed in Table 4.13.

Sample 4.36 *Deciphering the ACE AccessMask property for File System shares*

```
...:
...:
...:
248:
249:          Case cShareViaWMI, cShareViaADSI
250:               If (intACEMask And FILE_SHARE_FULL_ACCESS) Then
251:                  strTemp = strTemp & "," & "FILE_SHARE_FULL_ACCESS"
252:               End If
253:               If (intACEMask And FILE_SHARE_CHANGE_ACCESS) Then
254:                  strTemp = strTemp & "," & "FILE_SHARE_CHANGE_ACCESS"
255:               End If
256:               If (intACEMask And FILE_SHARE_READ_ACCESS) Then
257:                  strTemp = strTemp & "," & "FILE_SHARE_READ_ACCESS"
258:               End If
259:
...:
...:
...:
```

Based on that code, an execution of the script from the command line produces the following output if a File System share security is configured, as shown in Figure 4.21.

```
 1:  C:\>WMIManageSD.wsf /Share:MyDirectory /Decipher+
 2:  Microsoft (R) Windows Script Host Version 5.6
 3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:  Reading Share security descriptor via WMI from 'MyDirectory'.
 6:
 7:  +- Win32_SecurityDescriptor ----------------------------------------------------------------
 8:  | ControlFlags: ..... &h8004
 9:                                    SE_DACL_PRESENT
10:                                    SE_SELF_RELATIVE
11:  | DACL: ..............................'(Win32_ACE)
```

```
12:  | +- Win32_ACE ------------------------------------------------------------------------------
13:  | | AccessMask: ..... &h1F01FF
14:                          FILE_SHARE_FULL_ACCESS
15:                          FILE_SHARE_CHANGE_ACCESS
16:                          FILE_SHARE_READ_ACCESS
17:  | | AceFlags: ..... &h0
18:  | | AceType: ..... &h0
19:                          ACCESS_ALLOWED_ACE_TYPE
20:  | | Trustee: ..... (Win32_Trustee)
21:  | | +- Win32_Trustee ------------------------------------------------------------------------
22:  | | | Domain: ..... BUILTIN
23:  | | | Name: ..... Administrators
24:  | | | SID: ..... 1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0
25:  | | | SidLength: ..... 16
26:  | | | SIDString: ..... S-1-5-32-544
27:  | | +----------------------------------------------------------------------------------------
28:  | +--------------------------------------------------------------------------------------------
29:  | +- Win32_ACE ------------------------------------------------------------------------------
30:  | | AccessMask: ..... &h1200A9
31:                          FILE_SHARE_READ_ACCESS
32:  | | AceFlags: ..... &h0
33:  | | AceType: ..... &h0
34:                          ACCESS_ALLOWED_ACE_TYPE
35:  | | Trustee: ..... (Win32_Trustee)
36:  | | +- Win32_Trustee ------------------------------------------------------------------------
37:  | | | Name: ..... Everyone
38:  | | | SID: ..... 1,1,0,0,0,0,0,1,0,0,0,0
39:  | | | SidLength: ..... 12
40:  | | | SIDString: ..... S-1-1-0
41:  | | +----------------------------------------------------------------------------------------
42:  | +--------------------------------------------------------------------------------------------
43:  | +- Win32_ACE ------------------------------------------------------------------------------
44:  | | AccessMask: ..... &h1301BF
45:                          FILE_SHARE_CHANGE_ACCESS
46:                          FILE_SHARE_READ_ACCESS
47:  | | AceFlags: ..... &h0
48:  | | AceType: ..... &h0
49:                          ACCESS_ALLOWED_ACE_TYPE
50:  | | Trustee: ..... (Win32_Trustee)
51:  | | +- Win32_Trustee ------------------------------------------------------------------------
52:  | | | Domain: ..... LISSWARENET
53:  | | | Name: ..... MyGroup
54:  | | | SID: ..... 1,5,0,0,0,0,...,207,122,236,255,136,223,4,0,0
55:  | | | SidLength: ..... 28
56:  | | | SIDString: ..... S-1-5-21-3533506287-3489020660-2298473594-1247
57:  | | +----------------------------------------------------------------------------------------
58:  | +--------------------------------------------------------------------------------------------
59:  +----------------------------------------------------------------------------------------------
```

The *ACE Flags* property, although exposed by the WMI and the ADSI object model, is not applicable to a File System share, since the concept of inheritance does not exist for such a security descriptor type. This is why the value is always set to zero (lines 17, 32, and 47).

Figure 4.21
The File System share security descriptor user interface

To configure a security descriptor equivalent to the one shown in Figure 4.21, the following command lines must be used:

```
1:  C:\>WMIManageSD.wsf /Share:MyDirectory /Trustee:REMOVE_ALL_ACE /DelAce+
2:  C:\>WMIManageSD.wsf /Share:MyDirectory /Trustee:BUILTIN\Administrators
3:                                         /ACEType:ACCESS_ALLOWED_ACE_TYPE
4:                                         /ACEMask:FILE_SHARE_FULL_ACCESS,
5:                                                  FILE_SHARE_CHANGE_ACCESS,
6:                                                  FILE_SHARE_READ_ACCESS
7:                                         /ACEFlags:NONE /AddAce+
8:  C:\>WMIManageSD.wsf /Share:MyDirectory /Trustee:Everyone /DelAce+
9:  C:\>WMIManageSD.wsf /Share:MyDirectory /Trustee:LissWareNET\Everyone
10:                                        /ACEType:ACCESS_ALLOWED_ACE_TYPE
11:                                        /ACEMask:FILE_SHARE_READ_ACCESS
12:                                        /ACEFlags:NONE /AddAce+
13: C:\>WMIManageSD.wsf /Share:MyDirectory /Trustee:LissWareNET\MyGroup
14:                                        /ACEType:ACCESS_ALLOWED_ACE_TYPE
15:                                        /ACEType:ACCESS_ALLOWED_ACE_TYPE
16:                                        /ACEMask:FILE_SHARE_CHANGE_ACCESS,
17:                                                 FILE_SHARE_READ_ACCESS
18:                                        /ACEFlags:NONE /AddAce+
19: C:\>WMIManageSD.wsf /Share:MyDirectory /Decipher+
```

As for a folder, setting up the security descriptor of a File System share requires one execution per ACE configuration. Even if flag values are taken from Table 4.13, the logic is exactly the same as before. Note the *ACE Flags* set to "NONE," since inheritance is not supported for a File System share. As with a file or a folder, the WMI security descriptor access method is used, since no **/ADSI+** switch is specified.

4.11.4.5.3 The Active Directory object ACE AccessMask property

Managing the *ACE AccessMask* property of an Active Directory security descriptor is probably one of the most complex properties to handle. For Active Directory, we must first distinguish the standard rights from the Extended Rights. The standard rights are part of the system and cannot be modified.

However, because some directory-enabled applications may require the creation of some specific rights for the aim of the application, Active Directory offers a way to create new rights to protect Active Directory objects and attributes with more granularity. These rights are called the Active Directory Extended Rights and make use of the *ACE ObjectType* property. Of course, as we will see in section 4.11.4.5.3.2 ("Understanding the ACE InheritedObjectType property"), the use of an Extended Right is detected by also deciphering other ACE properties.

To decipher the standard Active Directory rights, the technique is still the same as before. A series of flags, defined in Tables 4.14 and 4.15, must be used to perform the deciphering bitwise operations.

Sample 4.37, which is part of the DecipherACEType() function, implements this logic. Due to the large number of rights, only a portion of the code is represented.

Sample 4.37 *Deciphering the ACE AccessMask property for Active Directory objects*

```
...:
...:
...:
259:
260:          Case cActiveDirectoryViaWMI, cActiveDirectoryViaADSI
261:               If (intACEMask = ADS_RIGHT_GENERIC_READ) Then
262:                  strTemp = strTemp & "," & "(ADS_RIGHT_GENERIC_READ)"
263:               End If
264:               If (intACEMask = ADS_RIGHT_GENERIC_WRITE) Then
265:                  strTemp = strTemp & "," & "(ADS_RIGHT_GENERIC_WRITE)"
266:               End If
267:               If (intACEMask = ADS_RIGHT_GENERIC_EXECUTE) Then
268:                  strTemp = strTemp & "," & "(ADS_RIGHT_GENERIC_EXECUTE)"
269:               End If
270:               If (intACEMask = ADS_RIGHT_GENERIC_ALL) Then
271:                  strTemp = strTemp & "," & "(ADS_RIGHT_GENERIC_ALL)"
272:               End If
...:
316:               If (intACEMask And ADS_RIGHT_DS_CONTROL_ACCESS) Then
317:                  strTemp = strTemp & "," & "ADS_RIGHT_DS_CONTROL_ACCESS"
318:               End If
319:
...:
...:
...:
```

Table 4.14 *The Active Directory Object ACE AccessMask Values—Standard View*

Granted & denied rights		Standard View																								
		Full Control	Read	Write	Create All Child Objects	Delete All Child Objects	Allowed to Authenticate	Change Password	Receive As	Reset Password	Send As	Read Account Restrictions	Write Account Restrictions	Read General Information	Write General Information	Read Group Membership	Write Group Membership	Read Logon Information	Write Logon Information	Read Personal Information	Write Personal Information	Read Phone and Mail Options	Write Phone and Mail Options	Read Public Information	Write Public Information
ACEType		**Standard**					**Extended**																			
ADS_ACETYPE_ACCESS_ALLOWED	0x0																									
ADS_ACETYPE_ACCESS_DENIED	0x1	X	X	X	X	X																				
ADS_ACETYPE_SYSTEM_AUDIT	0x2																									
ADS_ACETYPE_ACCESS_ALLOWED_OBJECT	0x5																									
ADS_ACETYPE_ACCESS_DENIED_OBJECT	0x6						X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ADS_ACETYPE_SYSTEM_AUDIT_OBJECT	0x7																									
ACEMask																										
ADS_RIGHT_GENERIC_ALL	0x10000000	X																								
ADS_RIGHT_GENERIC_EXECUTE	0x20000000																									
ADS_RIGHT_GENERIC_READ	0x80000000		X																							
ADS_RIGHT_GENERIC_WRITE	0x40000000			X																						
ADS_RIGHT_ACCESS_SYSTEM_SECURITY	0x1000000																									
ADS_RIGHT_ACTRL_DS_LIST	0x4	X	X																							
ADS_RIGHT_DELETE	0x10000	X																								
ADS_RIGHT_DS_CONTROL_ACCESS	0x100	X					X	X	X	X	X															
ADS_RIGHT_DS_CREATE_CHILD	0x1	X			X																					
ADS_RIGHT_DS_DELETE_CHILD	0x2	X				X																				
ADS_RIGHT_DS_DELETE_TREE	0x40	X																								
ADS_RIGHT_DS_LIST_OBJECT	0x80	X																								
ADS_RIGHT_DS_READ_PROP	0x10	X	X									X		X		X		X		X		X		X		
ADS_RIGHT_DS_SELF	0x8	X		X																						
ADS_RIGHT_DS_WRITE_PROP	0x20	X		X									X		X		X		X		X		X		X	
ADS_RIGHT_READ_CONTROL	0x20000	X	X	X	X	X																				
ADS_RIGHT_SYNCHRONIZE	0x100000																									
ADS_RIGHT_WRITE_DAC	0x40000	X																								
ADS_RIGHT_WRITE_OWNER	0x80000	X																								

Table 4.14 *The Active Directory Object ACE AccessMask Values—Standard View (continued)*

Granted & denied rights		Standard View																									
		Full Control	Read	Write	Create All Child Objects	Delete All Child Objects	Allowed to Authenticate	Change Password	Receive As	Reset Password	Send As	Read Account Restrictions	Write Account Restrictions	Read General Information	Write General Information	Read Group Membership	Write Group Membership	Read Logon Information	Write Logon Information	Read Personal Information	Write Personal Information	Read Phone and Mail Options	Write Phone and Mail Options	Read Public Information	Write Public Information
ObjectType		**Standard**					**Extended**																				
{68B1D179-0D15-4D4f-AB71-46152E79A7BC}							X																				
{AB721A53-1E2F-11D0-9819-00AA0040529B}								X																			
{AB721A56-1E2F-11D0-9819-00AA0040529B}									X																		
{00299570-246D-11D0-A768-00AA006E0529}										X																	
{AB721A54-1E2F-11D0-9819-00AA0040529B}											X																
{E45795B2-9455-11D1-AEBD-0000F80367C1}																						X					
{E45795B2-9455-11D1-AEBD-0000F80367C1}																							X				
{59BA2F42-79A2-11D0-9020-00C04FC2D3CF}														X													
{59BA2F42-79A2-11D0-9020-00C04FC2D3CF}															X												
{BC0AC240-79A9-11D0-9020-00C04FC2D4CF}																X											
{BC0AC240-79A9-11D0-9020-00C04FC2D4CF}																	X										
{77B5B886-944A-11D1-AEBD-0000F80367C1}																				X							
{77B5B886-944A-11D1-AEBD-0000F80367C1}																					X						
{E48D0154-BCF8-11D1-8702-00C04FB96050}																										X	
{E48D0154-BCF8-11D1-8702-00C04FB96050}																											X
{037088F8-0AE1-11D2-B422-00A0C968F939}																											
{037088F8-0AE1-11D2-B422-00A0C968F939}																											
{4C164200-20C0-11D0-A768-00AA006E0529}												X															
{4C164200-20C0-11D0-A768-00AA006E0529}													X														
{5F202010-79A5-11D0-9020-00C04FC2D4CF}																		X									
{5F202010-79A5-11D0-9020-00C04FC2D4CF}																			X								
{E45795B3-9455-11D1-AEBD-0000F80367C1}																											
{E45795B3-9455-11D1-AEBD-0000F80367C1}																											
ACEFlagType																											
ADS_FLAG_OBJECT_TYPE_PRESENT[1]	0x1						X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT[1]	0x2																										

(1) Only used when the ADSI object model is used to represent the security descriptor.

Table 4.15 *The Active Directory Object ACE AccessMask Values—Advanced View*

Granted & denied rights		Advanced View																
		Full Control	List Contents	Read All Properties	Write All Properties	Delete	Delete Subtree	Read Permissions	Modify Permissions	Modify Owner	All Validated Writes	All Extended Rights	Create All Child Objects	Delete All Child Objects	Allowed to authenticate	Change Password	Receive As	Reset Password	Send As
ACEType		**Standard**													**Extended**				
ADS_ACETYPE_ACCESS_ALLOWED	0x0																		
ADS_ACETYPE_ACCESS_DENIED	0x1	X	X	X	X	X	X	X	X	X	X	X	X	X					
ADS_ACETYPE_SYSTEM_AUDIT	0x2																		
ADS_ACETYPE_ACCESS_ALLOWED_OBJECT	0x5																		
ADS_ACETYPE_ACCESS_DENIED_OBJECT	0x6														X	X	X	X	X
ADS_ACETYPE_SYSTEM_AUDIT_OBJECT	0x7																		
ACEMask																			
ADS_RIGHT_GENERIC_ALL	0x10000000	X																	
ADS_RIGHT_GENERIC_EXECUTE	0x20000000																		
ADS_RIGHT_GENERIC_READ	0x80000000																		
ADS_RIGHT_GENERIC_WRITE	0x40000000																		
ADS_RIGHT_ACCESS_SYSTEM_SECURITY	0x1000000																		
ADS_RIGHT_ACTRL_DS_LIST	0x4	X	X												X	X	X	X	X
ADS_RIGHT_DELETE	0x10000	X				X													
ADS_RIGHT_DS_CONTROL_ACCESS	0x100	X										X							
ADS_RIGHT_DS_CREATE_CHILD	0x1	X											X						
ADS_RIGHT_DS_DELETE_CHILD	0x2	X												X					
ADS_RIGHT_DS_DELETE_TREE	0x40	X					X												
ADS_RIGHT_DS_LIST_OBJECT	0x80	X																	
ADS_RIGHT_DS_READ_PROP	0x10	X		X															
ADS_RIGHT_DS_SELF	0x8	X									X								
ADS_RIGHT_DS_WRITE_PROP	0x20	X			X														
ADS_RIGHT_READ_CONTROL	0x20000	X						X											
ADS_RIGHT_SYNCHRONIZE	0x100000																		
ADS_RIGHT_WRITE_DAC	0x40000	X							X										
ADS_RIGHT_WRITE_OWNER	0x80000	X								X									
ObjectType																			
{68B1D179-0D15-4D4f-AB71-46152E79A7BC}															X				
{AB721A53-1E2F-11D0-9819-00AA0040529B}																X			
{AB721A56-1E2F-11D0-9819-00AA0040529B}																	X		
{00299570-246D-11D0-A768-00AA006E0529}																		X	
{AB721A54-1E2F-11D0-9819-00AA0040529B}																			X
ACEFlagType																			
ADS_FLAG_OBJECT_TYPE_PRESENT[1]	0x1														X	X	X	X	X
ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT[1]	0x2																		

(1) Only used when the ADSI object model is used to represent the security descriptor.

Active Directory objects can inherit an ACE from parent objects. Therefore, some flags define how the ACE inheritance behaves. These flags are summarized in Table 4.16.

When configuring ACE inheritance, it is possible to specify that the ACE inheritance will only apply on a particular Active Directory object class. This inheritance makes use of the *ACE InheritedObjectType* property, which contains a GUID number. To understand how to set up the *ACE InheritedObjectType* property, it is best to understand how Active Directory Extended Rights work first. This will help us to discover the links that exist between Active Directory classes, Active Directory attributes, and Active Directory rights. It is important to note that Active Directory Extended

Table 4.16 *The Active Directory Objects ACE Flags Values*

Inheritance & Audit		This object only	This object and all child objects	Child objects only	Audit Successful access	Audit Failed access
ACEFlags						
NONE	0x0	X				
ADS_ACEFLAG_INHERIT_ACE	0x2		X	X		
ADS_ACEFLAG_INHERIT_ONLY_ACE	0x8			X		
ADS_ACEFLAG_INHERITED_ACE[1]	0x10					
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	0x4					
ADS_ACEFLAG_VALID_INHERIT_FLAGS[1]	0x1F					
ADS_ACEFLAG_SUCCESSFUL_ACCESS	0x40				X	
ADS_ACEFLAG_FAILED_ACCESS	0x80					X

(1) can only be set by the system.

Rights and ACE inheritance on specific object classes are two different things, but they use the same type of information for their configuration. This explains why it is easier to discover the Active Directory Extended Rights first.

4.11.4.5.3.1 Understanding the ACE ObjectType property Having a good knowledge of the different Naming Contexts that Active Directory implements and how they are structured is important with respect to the origin of the GUID number contained in the *ACE ObjectType* property.

All Active Directory objects support a standard set of access rights, listed in Table 4.14. You can use these access rights in the *ACE AccessMask* of an object's security descriptor to control access at the object level. However, some objects' classes may require an access control not supported by the standard access rights. In such a case, Active Directory allows you to extend the standard access control mechanism to perform a more granular control on some Active Directory objects and attributes. An Active Directory Extended Right is an Active Directory object created from the **controlAccessRight** object class. All Active Directory Extended Rights are located in the "CN=Extended-Rights" container of the Active Directory Configuration naming context. To correctly decipher an ACE of an Active Directory Extended Right, we must distinguish between different Extended Rights types. The type is determined by the **validAccesses** attribute value defined in each **controlAccessRight** object created in the "CN=Extended-Rights" container. There are three Extended Rights types:

- **The Extended Rights enforced by Active Directory:** These Extended Rights are enforced by Active Directory to grant (or deny) a read or write operation to an Active Directory property set. A **validAccesses** attribute value of 0x30 (ADS_RIGHT_DS_READ_PROP or ADS_RIGHT_DS_WRITE_PROP) defines this type of Extended Rights. They have an *ACE Type* set to one of these three values: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, ADS_ACETYPE_ACCESS_DENIED_OBJECT, or ADS_ACETYPE_SYSTEM_AUDIT_OBJECT. The *ACE AccessMask* value is equal to a logical combination of the ADS_RIGHT_DS_READ_PROP and ADS_RIGHT_DS_WRITE flags (see Table 4.14).
- **The Extended Rights enforced by applications:** These rights are enforced by applications, which could be, for instance, Exchange 2000, Outlook 2000, or the system itself but not Active Directory. A **validAccesses** attribute value of 0x100 (ADS_RIGHT_DS_CONTROL_ACCESS) defines this type of Extended Rights. They have an *ACE Type* set to one of these three values: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, ADS_ACETYPE_ACCESS_DENIED_OBJECT, or ADS_ACETYPE_SYSTEM_AUDIT_OBJECT. They have an *ACE AccessMask* value equal to the ADS_RIGHT_DS_CONTROL_ACCESS flag value (see Table 4.14).
- **The Extended Rights enforced by the system to perform extra checking:** These Extended Rights are called the "Validated Writes." These rights are used by the system to perform a value check or validation before writing a value to a property on an object. The value checking or validation goes beyond what is required by the Active Directory schema. This type of right uses a value of 0x8 (ADS_RIGHT_DS_SELF) in the **validAccesses** attribute. They have an *ACE Type* set to one of these three values: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, ADS_ACETYPE_ACCESS_DENIED_OBJECT, or ADS_ACETYPE_SYSTEM_AUDIT_OBJECT. They have an *ACE AccessMask* value equal to the ADS_RIGHT_DS_SELF flag value.

Figure 4.22 shows an example of the three Extended Rights types. On the left, we have the "Personal Information" right, which is enforced by Active Directory (**validAccesses** = 0x30). In the center, we have the "Send As" right, which is enforced by an application (**validAccesses** = 0x100). On the right, we have the "Add/Remove self as member" right, which is only enforced by the system itself (**validAccesses** = 0x08).

To continue, we will use the three rights in Figure 4.22 as examples. Because rights are always related to an Active Directory object, Extended

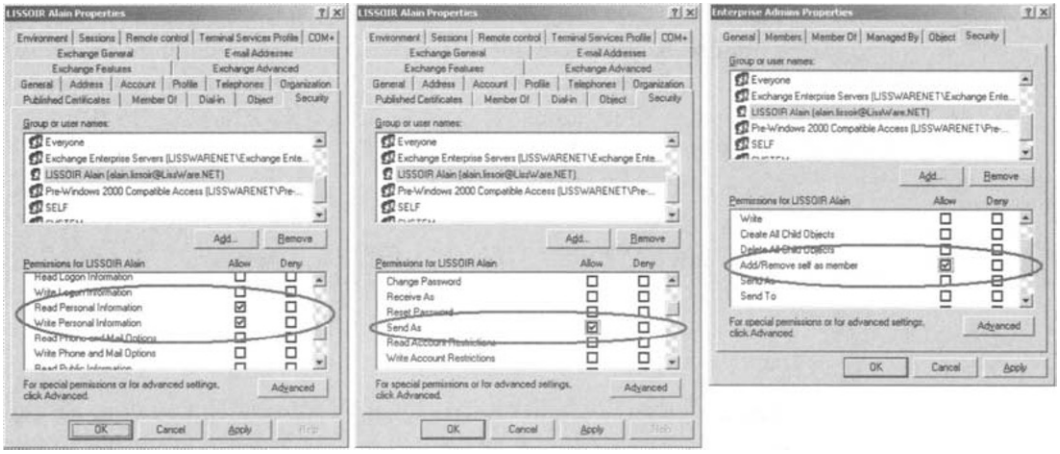


Figure 4.22 *The Extended Rights enforced by Active Directory (left), enforced by applications (center), and enforced by the system (right).*

Rights have a link with the Active Directory object classes they apply to. For instance, the Extended Rights in Figure 4.22 ("Personal Information" and "Send As") are linked with the Active Directory **user** class defined in the Active Directory schema, because they apply to objects created from the **user** class. The same rule applies for the "Add/Remove self as member" right, but it is linked with the **group** class. The link between the Extended right and the **user** class or the **group** class is made with an attribute available from the **controlAccessRight** object, called the **appliesTo** attribute. The **appliesTo** attribute may contain one or more GUID numbers, where each GUID number is the value contained in the **schemaIDGUID** attribute of the class that the Extended Rights relates to. For instance, the "Personal Information" Extended Right has several GUID numbers in the **appliesTo** attribute (Figure 4.23, left pane), where each of them is coming from the **schemaIDGUID** attribute of the corresponding classes (Figure 4.23, right pane) for the **user** class.

Although the format of the GUID number in the **schemaIDGUID** attribute is in binary, it is the same GUID number. Figure 4.24 illustrates the logic to use to convert a binary GUID number to a string GUID number and vice versa.

If we look for the same information for the "Send As" Extended Right, we find the same type of relationship (see Figure 4.25). The same rule will apply if you look at the GUID number stored in the **schemaIDGUID** attribute of the **group** class.

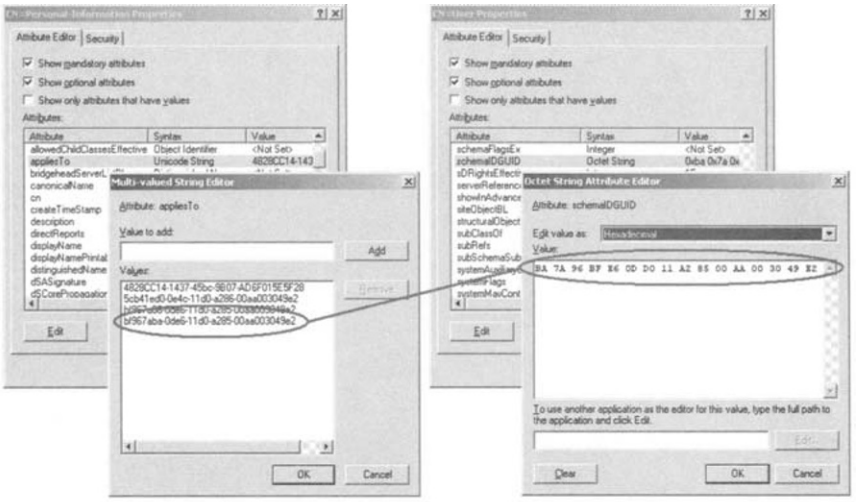


Figure 4.23 *The appliesTo GUID numbers of the "Personal Information" Extended Right in liaison with the schemaIDGUID attribute of the classSchema object.*

The aim of the "Personal Information" Extended Right (and of all Extended Rights using a **validAccesses** attribute value equal to 0x30) is to protect some Active Directory attributes associated with the class that the Extended Right refers to. This means that a relationship between Extended Rights and some Active Directory attributes also exists. To establish the link between an Extended Right and the set of attributes it protects, an Extended Right of this type (**validAccesses** = 0x30) uses another GUID number, which is stored in the **rightsGUID** attribute of the **controlAccessRight** object. Any attributes that can be protected by the Extended Right refer to the Extended Right GUID number by storing the value in its **attributeSecurityGUID** attribute (see Figure 4.26). The **attributeSecuri-**

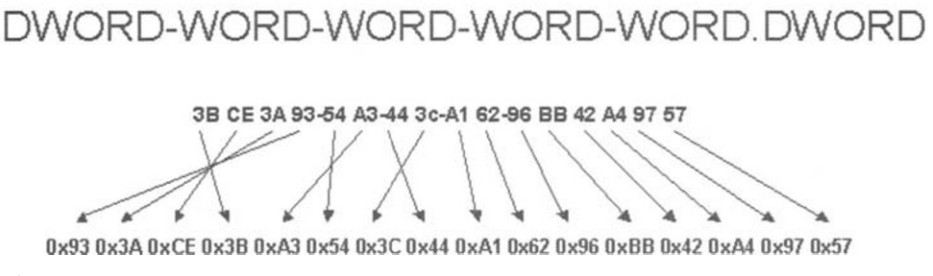


Figure 4.24 *Converting a GUID string to a GUID number and vice versa.*

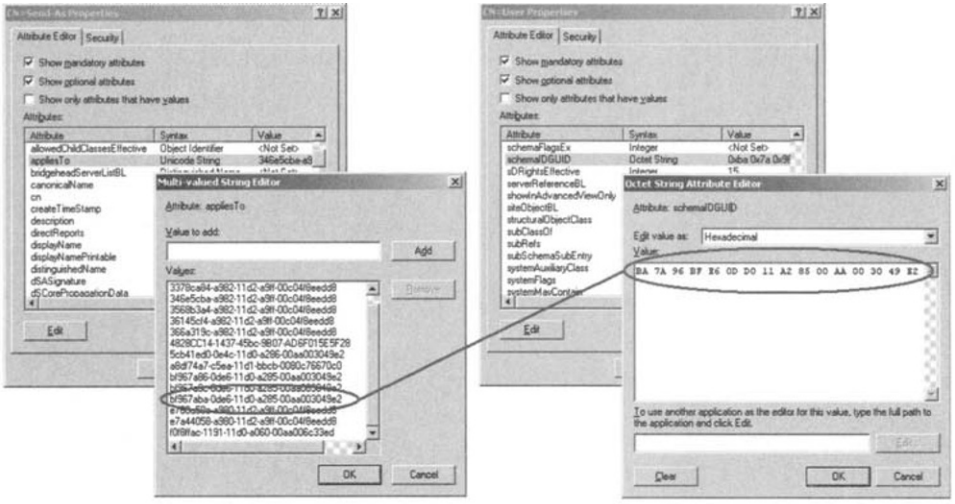


Figure 4.25 *The appliesTo GUID numbers of the "Send As" Extended Right in liaison with the schemaIDGUID attribute of the classSchema object.*

tyGUID attribute is part of the **attributeSchema** object defining the attribute in the Active Directory Schema.

Finally, Figure 4.27 summarizes the links between Extended Rights (**controlAccessRight**), Active Directory object classes (**classSchema**), and

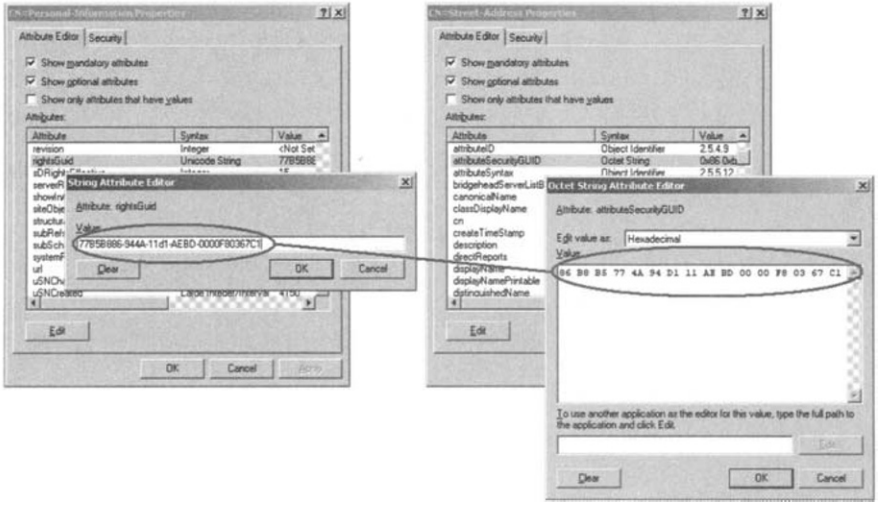


Figure 4.26 *The attributeSecurityGUID attribute of the attributeSchema object contains the rightsGUID GUID number of the "Personal Information" Extended Right.*

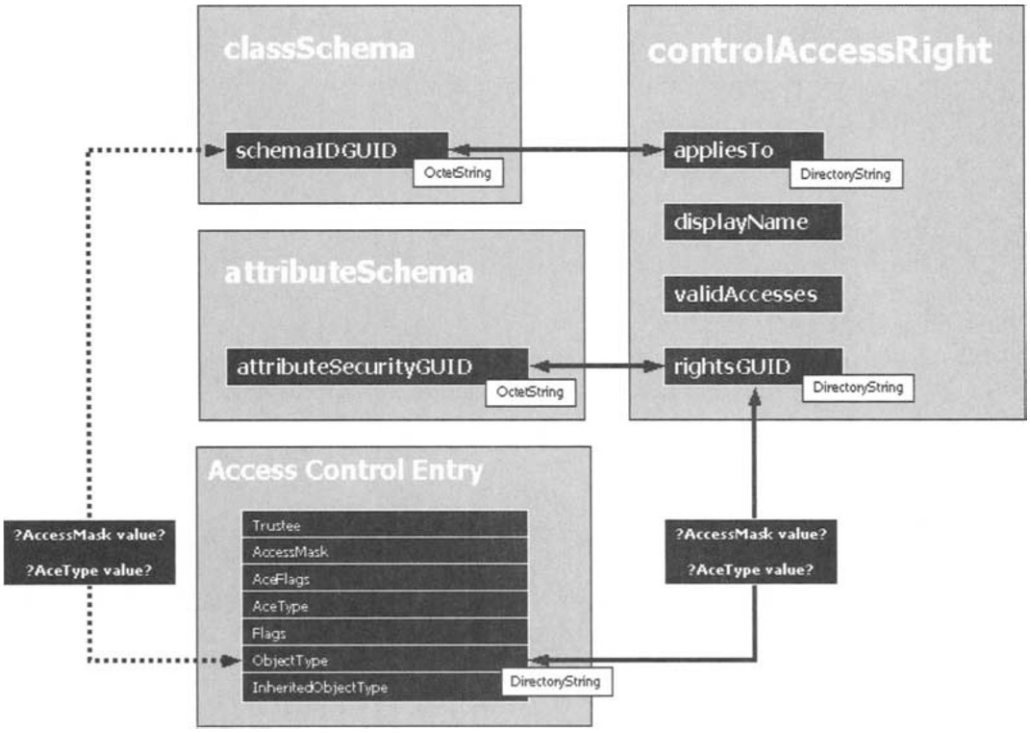


Figure 4.27 *The Extended Rights attributes links.*

attribute definitions (**attributeSchema**). Figure 4.27 also shows that the *ACE ObjectType* property of a security descriptor using an Extended Right refers to the GUID number value stored in the **rightsGUID** attribute of the **controlAccessRight** object. However, we will see later in this section that the GUID number could come from the **schemaIDGUID** attribute of a **classSchema** object. This is why we have a dashed line in Figure 4.27.

When Extended Rights use a **validAccesses** attribute value of 0x100, they do not refer to any particular attribute, since this type of right is enforced at the application level, which means that it is the responsibility of the application to validate the right (i.e., Exchange 2000 and Outlook 2000 validate the "Send As" Extended Right). The last type (**validAccesses** = 0x8) is enforced by the system and is used to lock write operations to some Active Directory attributes (i.e., "Validated write to DNS host name" right applying to the **computer** class). Table 4.17 summarizes the Extended Rights names and GUID numbers available under Windows Server 2003 and Exchange 2000 SP3 with the classes and attributes they apply to.

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Account Restrictions	{4c164200-20c0-11d0-a768-00aa006e0529}	0x30	inetOrgPerson computer user	accountExpires msDS-User-Account-Control-Computed pwdLastSet userAccountControl userParameters
Add GUID	{440820ad-65b4-11d1-a3da-0000f875ae0d}	0x100	domainDNS	
Add PF to admin group	{ce4c81a8-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchAdminGroup	
Add/Remove Replica In Domain	{9923a32a-3607-11d2-b9be-0000f87a36b2}	0x100	domainDNS	
Add/Remove self as member	{bf9679c0-0de6-11d0-a285-00aa003049e2}	0x8	group	
Administer information store	{d74a8762-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchStorageGroup msExchServersContainer msExchPublicMDB msExchPseudoPFAdmin msExchPrivateMDB msExchPFTree msExchOrganizationContainer msExchExchangeServer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Allocate Rids	{1abd7cf8-0a99-11d1-adbb-00c04fd8d5cd}	0x100	nTDSDSA	
Allowed to Authenticate	{68b1d179-0d15-4d4f-ab71-46152e79a7bc}	0x100	inetOrgPerson user computer	
Apply Group Policy	{edacfd8f-ffb3-11d1-b41d-00a0c968f939}	0x100	groupPolicyContainer	
Change Domain Master	{014bf69c-7b3b-11d1-85f6-08002be74fab}	0x100	crossRefContainer	
Change Infrastructure Master	{cc17b1fb-33d9-11d2-97d4-00c04fd8d5cd}	0x100	infrastructureUpdate	
Change Password	{ab721a53-1e2f-11d0-9819-00aa0040529b}	0x100	inetOrgPerson computer user	
Change PDC	{bae50096-4752-11d1-9052-00c04fc2d4cf}	0x100	domainDNS	
Change Rid Master	{d58d5f36-0a98-11d1-adbb-00c04fd8d5cd}	0x100	rIDManager	
Change Schema Master	{e12b56b6-0a95-11d1-adbb-00c04fd8d5cd}	0x100	dMD	
Check Stale Phantoms	{69ae6200-7f46-11d2-b9ad-00c04f79f805}	0x100	nTDSDSA	
Create Inbound Forest Trust	{e2a36dc9-ae17-47c3-b58b-be34c55ba633}	0x100	domainDNS	
Create named properties in the information store	{d74a8766-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchStorageGroup msExchServersContainer msExchPublicMDB msExchPrivateMDB msExchPFTree msExchOrganizationContainer msExchExchangeServer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Create public folder	{cf0b3dc8-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPFTree msExchOrganizationContainer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Create top level public folder	{cf4b9d46-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPFTree msExchOrganizationContainer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
DNS Host Name Attributes	{72e39547-7b18-11d1-adef-00c04fd8d5cd}	0x30	computer	dNSHostName msDS-AdditionalDnsHostName
Do Garbage Collection	{fec364e0-0a98-11d1-adbb-00c04fd8d5cd}	0x100	nTDSDSA	
Domain Administer Server	{ab721a52-1e2f-11d0-9819-00aa0040529b}	0x100	samServer	
Domain Password & Lockout Policies	{c7407360-20bf-11d0-a768-00aa006e0529}	0x30	domainDNS domain	lockOutObservationWindow lockoutDuration lockoutThreshold maxPwdAge minPwdAge minPwdLength pwdHistoryLength pwdProperties
Enable Per User Reversibly Encrypted Password	{05c74c5e-4deb-43b4-bd9f-86664c2a7fd5}	0x100	domainDNS	
Enroll	{0e10c968-78fb-11d2-90d4-00c04f79dc55}	0x100	pKICertificateTemplate	
Enumerate Entire SAM Domain	{91d67418-0135-4acc-8d79-c08e857cfbec}	0x100	samServer	
Exchange administrator	{8e48d5a8-b09e-11d2-aa06-00c04f8eedd8}	0x100	msExchAdminGroup	
Exchange full administrator	{8e6571e0-b09e-11d2-aa06-00c04f8eedd8}	0x100	msExchAdminGroup	
Exchange public folder read-only administrator	{8ff1383c-b09e-11d2-aa06-00c04f8eedd8}	0x100	msExchAdminGroup	
Exchange public folder service	{90280e52-b09e-11d2-aa06-00c04f8eedd8}	0x100	msExchAdminGroup	
Execute Forest Update Script	{2f16c4a5-b98e-432c-952a-cb388ba33f2e}	0x100	crossRefContainer	
General Information	{59ba2f42-79a2-11d0-9020-00c04fc2d3cf}	0x30	inetOrgPerson user	adminDescription codePage countryCode displayName objectSid primaryGroupID sAMAccountName sAMAccountType sDRightsEffective showInAdvancedViewOnly sIDHistory uid comment

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Generate Resultant Set of Policy (Logging)	{b7b1b3de-ab09-4242-9e30-9980e5d322f7}	0x100	domainDNS organizationalUnit	
Generate Resultant Set of Policy (Planning)	{b7b1b3dd-ab09-4242-9e30-9980e5d322f7}	0x100	domainDNS organizationalUnit	
Group Membership	{bc0ac240-79a9-11d0-9020-00c04fc2d4cf}	0x30	inetOrgPerson user	memberOf member
Logon Information	{5f202010-79a5-11d0-9020-00c04fc2d4cf}	0x30	inetOrgPerson user	badPwdCount homeDirectory homeDrive lastLogoff lastLogon lastLogonTimestamp logonCount logonHours logonWorkstation profilePath scriptPath userWorkstations
Mail-enable public folder	{cf899a6a-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPFTree msExchOrganizationContainer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Manage Replication Topology	{1131f6ac-9c07-11d1-f79f-00c04fc2dcd2}	0x100	dMD configuration domainDNS	
Migrate SID History	{ba33815a-4f93-4c76-87f3-57574bff8109}	0x100	domainDNS	
Modify public folder ACL	{d74a8769-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchPseudoPFAdmin msExchPFTree	
Modify public folder admin ACL	{d74a876f-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchPseudoPFAdmin msExchPFTree msExchOrganizationContainer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Modify public folder deleted item retention	{cffe6da4-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPseudoPFAdmin msExchPFTree msExchAdminGroup	
Modify public folder expiry	{cfc7978e-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPseudoPFAdmin msExchPFTree msExchAdminGroup	
Modify public folder quotas	{d03a086e-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchPseudoPFAdmin msExchPFTree msExchAdminGroup	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Modify public folder replica list	{d0780592-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchStorageGroup msExchServersContainer msExchPublicMDB msExchPseudoPFAdmin msExchPFTree msExchOrganizationContainer msExchExchangeServer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Monitor Active Directory Replication	{f98340fb-7c5b-4cdb-a00b-2ebdfa115a96}	0x100	dMD configuration domainDNS	
Open Address List	{a1990816-4298-11d1-ade2-00c04fd8d5cd}	0x100	addressBookContainer	
Open Connector Queue	{b4e60130-df3f-11d1-9c86-006008764d0e}	0x100	site	
Open mail send queue	{d74a8774-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchStorageGroup msExchServersContainer msExchPublicMDB msExchPrivateMDB msExchOrganizationContainer msExchExchangeServer msExchAdminGroupContainer msExchAdminGroup	
Other Domain Parameters (for use by SAM)	{b8119fd0-04f6-4762-ab7a-4986c76b3f9a}	0x30	domainDNS	domainReplica forceLogoff modifiedCount oEMInformation serverRole serverState uASCompat
Peek Computer Journal	{4b6e08c3-df3c-11d1-9c86-006008764d0e}	0x100	mSMQConfiguration	
Peek Dead Letter	{4b6e08c1-df3c-11d1-9c86-006008764d0e}	0x100	mSMQConfiguration	
Peek Message	{06bd3201-df3e-11d1-9c86-006008764d0e}	0x100	mSMQQueue	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Personal Information	{77b5b886-944a-11d1-aebd-0000f80367c1}	0x30	inetOrgPerson computer contact user	streetAddress homePostalAddress assistant info c facsimileTelephoneNumber internationalISDNNumber l publicDelegates mSMQDigests mSMQSignCertificates personalTitle otherFacsimileTelephoneNumber otherHomePhone homePhone otherIpPhone ipPhone primaryInternationalISDNNumber otherMobile mobile otherTelephone otherPager pager physicalDeliveryOfficeName thumbnailPhoto postOfficeBox postalAddress postalCode preferredDeliveryMethod registeredAddress st street telephoneNumber teletexTerminalIdentifier telexNumber primaryTelexNumber userCert userSharedFolder userSharedFolderOther userSMIMECertificate x121Address userCertificate
Phone and Mail Options	{e45795b2-9455-11d1-aebd-0000f80367c1}	0x30	inetOrgPerson group user	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Public Information	{e48d0154-bcf8-11d1-8702-00c04fb96050}	0x30	inetOrgPerson computer user	notes allowedAttributes allowedAttributesEffective allowedChildClasses allowedChildClassesEffective altSecurityIdentities cn company department d escription displayNamePrintable division mail givenName initials legacyExchangeDN manager msDS-AllowedToDelega
Read metabase properties	{be013017-13a1-41ad-a058-f156504cb617}	0x100	msExchServersContainer protocolCfgSharedServer msExchOrganizationContainer msExchExchangeServer msExchAdminGroupContainer msExchAdminGroup	
Reanimate Tombstones	{45ec5156-db7e-47bb-b53f-dbeb2d03c40f}	0x100	dMD configuration domainDNS	
Recalculate Hierarchy	{0bc1554e-0a99-11d1-adbb-00c04fd8d5cd}	0x100	nTDSDSA	
Recalculate Security Inheritance	{62dd28a8-7f46-11d2-b9ad-00c04f79f805}	0x100	nTDSDSA	
Receive As	{ab721a56-1e2f-11d0-9819-00aa0040529b}	0x100	msExchServersContainer msExchPublicMDB protocolCfgSMTPServer msExchPrivateMDB msExchOrganizationContainer mTA msExchExchangeServer msExchAdminGroupContainer msExchAdminGroup inetOrgPerson computer user	
Receive Computer Journal	{4b6e08c2-df3c-11d1-9c86-006008764d0e}	0x100	mSMQConfiguration	
Receive Dead Letter	{4b6e08c0-df3c-11d1-9c86-006008764d0e}	0x100	mSMQConfiguration	
Receive Journal	{06bd3203-df3e-11d1-9c86-006008764d0e}	0x100	mSMQQueue	
Receive Message	{06bd3200-df3e-11d1-9c86-006008764d0e}	0x100	mSMQQueue	
Refresh Group Cache for Logons	{9432c620-033c-4db7-8b58-14ef6d0bf477}	0x100	nTDSDSA	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Remote Access Information	{037088f8-0ae1-11d2-b422-00a0c968f939}	0x30	inetOrgPerson user	msNPAllowDialin msNPCallingStationID msRADIUSCallbackNumber msRADIUSFramedIPAddress msRADIUSFramedRoute msRADIUSServiceType tokenGroups tokenGroupsGlobalAndUniversal tokenGroupsNoGCAcceptable
Remove PF from admin group	{d0b86510-afe6-11d2-aa04-00c04f8eedd8}	0x100	msExchAdminGroup	
Replicating Directory Changes	{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}	0x100	dMD configuration domainDNS	
Replicating Directory Changes All	{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}	0x100	dMD configuration domainDNS	
Replication Synchronization	{1131f6ab-9c07-11d1-f79f-00c04fc2dcd2}	0x100	dMD configuration domainDNS	
Reset Password	{00299570-246d-11d0-a768-00aa006e0529}	0x100	inetOrgPerson computer user	
Send As	{ab721a54-1e2f-11d0-9819-00aa0040529b}	0x100	msExchServersContainer msExchPublicMDB publicFolder protocolCfgSMTPServer msExchPrivateMDB msExchOrganizationContainer mTA msExchExchangeServer msExchAdminGroupContainer msExchAdminGroup group contact inetOrgPerson computer user	

Table 4.17 *Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included) (continued)*

Extended Right Display Name	GUID number (rightsGUID)	Type	Related classes	Protected attributes
Send Message	{06bd3202-df3e-11d1-9c86-006008764d0e}	0x100	msMQ-Group msMQQueue	
Send To	{ab721a55-1e2f-11d0-9819-00aa0040529b}	0x100	group	
Unexpire Password	{ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501}	0x100	domainDNS	
Update Password Not Required Bit	{280f369c-67c7-438e-ae98-1d46f3c6f541}	0x100	domainDNS	
Update Schema Cache	{be2bb760-7f46-11d2-b9ad-00c04f79f805}	0x100	dMD	
Validated write to DNS host name	{72e39547-7b18-11d1-adef-00c04fd8d5cd}	0x8	computer	dNSHostName msDS-AdditionalDnsHostName
Validated write to service principal name	{f3a64788-5306-11d1-a9c5-0000f80367c1}	0x8	computer	
View information store status	{d74a875e-22b9-11d3-aa62-00c04f8eedd8}	0x100	msExchStorageGroup msExchServersContainer msExchPublicMDB msExchPseudoPFAdmin msExchPrivateMDB msExchPFTree msExchOrganizationContainer msExchExchangeServer msExchConfigurationContainer msExchAdminGroupContainer msExchAdminGroup	
Web Information	{e45795b3-9455-11d1-aebd-0000f80367c1}	0x30	inetOrgPerson contact user	wWWHomePage url

Each time, an "ACE" refers to an Extended Right, the *ACE ObjectType* GUID number must be searched in this table to find the corresponding Extended Rights name. Table 4.17 can also be used to determine which Extended Right protects a specific attribute of a specific Active Directory object. This should ease the process of determining which right must be set to secure a specific attribute. For instance, based on Table 4.17, we know that the "Personal Information" Extended Right protects the **street** and **telephoneNumber** attributes, among others.

To understand how to decipher this type of ACE, let's take examples from Figure 4.22. We see that a user called "LISSOIR Alain" is granted to read and change his personal information (left). At the same time, he is also granted the "Send As" right (center), and he can add or remove himself from the "Enterprise Admins" group.

By using the **WMIManageSD.Wsf** script with the following command line, deciphering this Active Directory security descriptor produces the following output:

```
  1:  C:\>WMIManageSD.Wsf /ADObject:"CN=LISSOIR Alain,CN=Users,DC=..." /Decipher+ /ADSI+
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:  Reading AD object security descriptor via ADSI from 'LDAP://CN=LISSOIR Alain,CN=Users,DC=...
  6:
  7:  +- ADSI Security Descriptor ------------------------------------------------------------------
  8:  | Owner: ..... LISSWARENET\Domain Admins
  9:  | Group: ..... LISSWARENET\Domain Admins
 10:  | Revision: ..... 1
 11:  | Control: ..... &h9C14
 ..:
 18:  |+- ADSI DiscretionaryAcl -------------------------------------------------------------------
 19:  ||+- ADSI ACE ---------------------------------------------------------------------------------
 ..:
 38:  ||+----------------------------------------------------------------------------------------
...:
186:  ||+- ADSI ACE ---------------------------------------------------------------------------------
187:  ||| AccessMask: ..... &h100
188:                                        ADS_RIGHT_DS_CONTROL_ACCESS
189:  ||| AceFlags: ..... &h0
190:  ||| AceType: ..... &h5
191:                                        ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
192:  ||| AceFlagType: ..... &h1
193:                                        ADS_FLAG_OBJECT_TYPE_PRESENT
194:  ||| ObjectType: ..... {AB721A54-1E2F-11D0-9819-00AA0040529B}
195:  ||| Trustee: ..... LISSWARENET\Alain.Lissoir
196:  ||+----------------------------------------------------------------------------------------
197:  ||+- ADSI ACE ---------------------------------------------------------------------------------
198:  ||| AccessMask: ..... &h30
199:                                        ADS_RIGHT_DS_READ_PROP
200:                                        ADS_RIGHT_DS_WRITE_PROP
201:  ||| AceFlags: ..... &h0
202:  ||| AceType: ..... &h5
```

```
203:                                          ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
204:   ||| AceFlagType: ..................... &h1
205:                                          ADS_FLAG_OBJECT_TYPE_PRESENT
206:   ||| ObjectType: ..................... {77B5B886-944A-11D1-AEBD-0000F80367C1}
207:   ||| Trustee: ......................... LISSWARENET\Alain.Lissoir
208:   ||+-------------------------------------------------------------------------------------
...:
232:   |+--------------------------------------------------------------------------------------
233:   +---------------------------------------------------------------------------------------
...:
...:
...:
```

From line 186 through 196, the "Send As" Extended Right is granted to trustee "Alain.Lissoir" as:

- The *ACE Type* has a value equal to ADS_ACETYPE_ACCESS_ALLOWED_OBJECT (line 191).
- The *ACE AccessMask* has a value equal to ADS_RIGHT_DS_CONTROL_ACCESS ACE (line 188).
- The *ACE ObjectType* property has a GUID number corresponding to the "Send As" Extended Right (line 194). Check Table 4.17 to find the Extended Right GUID number with its corresponding display name.

In the same way, from line 197 through 208, the "Personal Information" Extended Right is granted to trustee "Alain.Lissoir" to read and write the personal information as:

- The *ACE Type* has a value equal to ADS_ACETYPE_ACCESS_ALLOWED_OBJECT (line 203).
- The *ACE AccessMask* has a value equal to ADS_RIGHT_DS_READ_PROP + ADS_RIGHT_DS_WRITE_PROP (lines 199 and 200).
- The *ACE ObjectType* property has a GUID number corresponding to the "Personal Information" Extended Right (line 206). Check Table 4.17 to find the Extended Right GUID number with its corresponding display name.

For the "Add/Remove self as member" Extended Right, the same logic applies with different values.

```
1:  C:\>WMIManageSD.Wsf /ADObject:"CN=Enterprise Admins,CN=Users,DC=..." /Decipher+ /ADSI+
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Reading AD object security descriptor via ADSI from 'LDAP://CN=Enterprise Admins,CN=...'.
6:
```

```
  7:  +- ADSI Security Descriptor ------------------------------------------------------------------
  8:  | Owner: ............................... LISSWARENET\Domain Admins
  9:  | Group: ............................... LISSWARENET\Domain Admins
 10:  | Revision: ............................ 1
 11:  | Control: ............................. &h9C14
 ..:
 18:  |+- ADSI DiscretionaryAcl --------------------------------------------------------------------
 19:  ||+- ADSI ACE ----------------------------------------------------------------------------------
 ..:
 38:  ||+----------------------------------------------------------------------------------------------
...:
174:  ||+- ADSI ACE ----------------------------------------------------------------------------------
175:  ||| AccessMask: ......................... &h8
176:                                            ADS_RIGHT_DS_SELF
177:  ||| AceFlags: ........................... &h2
178:                                            ADS_ACEFLAG_CONTAINER_INHERIT_ACE
179:                                            ADS_ACEFLAG_VALID_INHERIT_FLAGS
180:  ||| AceType: ........................... &h5
181:                                            ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
182:  ||| AceFlagType: ....................... &h1
183:                                            ADS_FLAG_OBJECT_TYPE_PRESENT
184:  ||| ObjectType: ....................... {BF9679C0-0DE6-11D0-A285-00AA003049E2}
185:  ||| Trustee: .......................... LISSWARENET\Alain.Lissoir
186:  ||+----------------------------------------------------------------------------------------------
...:
224:  |+-----------------------------------------------------------------------------------------------
225:  +------------------------------------------------------------------------------------------------
```

From line 174 through 186, the right "Add/Remove self as member" is granted to trustee "Alain.Lissoir" as:

- The *ACE Type* has a value equal to ADS_ACETYPE_ACCESS_ALLOWED_OBJECT (line 181).
- The *ACE AccessMask* has a value equal to ADS_RIGHT_DS_SELF (line 176).
- The *ACE ObjectType* property has a GUID number corresponding to the "Add/Remove self as member" Extended Right (line 184). Check Table 4.17 to find the Extended Right GUID number with its corresponding display name.

When the *ACE AccessMask* property has a value coming from a combination of the ADS_RIGHT_DS_CREATE_CHILD and ADS_RIGHT_DS_DELETE_CHILD flags, the *ACE ObjectType* property contains a GUID number, but it does not refer to an Extended Right. In this case, the GUID number refers to the **schemaIDGUID** of an Active Directory **classSchema** object and defines a permission that grants or denies a trustee the right to create or delete objects of the referred class type. The dashed line in Figure 4.27 represents this link. Figure 4.28 shows an example of such a right on the "CN=Users" container.

If we decipher the ACE, we obtain the following result:

```
  1:  C:\>WMIManageSD.Wsf /ADObject:"CN=Users,DC=LissWare,DC=Net" /Decipher+ /ADSI+
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:  Reading AD object security descriptor via ADSI from 'LDAP://CN=Users,DC=LissWare,DC=Net'.
  6:
  7:  +- ADSI Security Descriptor ----------------------------------------------------------------
  8:  | Owner: ..................................... LISSWARENET\Domain Admins
  9:  | Group: ..................................... LISSWARENET\Domain Admins
 10:  | Revision: .................................. 1
 11:  | Control: ................................... &h8C14
 ..:
 17:  |+- ADSI DiscretionaryAcl -----------------------------------------------------------------
 18:  ||+- ADSI ACE ------------------------------------------------------------------------------
 ..:
 29:  ||+-------------------------------------------------------------------------------------------
...:
105:  ||+- ADSI ACE ------------------------------------------------------------------------------
106:  ||| AccessMask: ................................ &h3
107:                                                   ADS_RIGHT_DS_CREATE_CHILD
108:                                                   ADS_RIGHT_DS_DELETE_CHILD
109:  ||| AceFlags: ................................. &h2
110:                                                   ADS_ACEFLAG_CONTAINER_INHERIT_ACE
111:                                                   ADS_ACEFLAG_VALID_INHERIT_FLAGS
112:  ||| AceType: ................................. &h5
113:                                                   ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
114:  ||| AceFlagType: ............................. &h1
115:                                                   ADS_FLAG_OBJECT_TYPE_PRESENT
116:  ||| ObjectType: ............................. {35BE884C-A982-11D2-A9FF-00C04F8EEDD8}
117:  ||| Trustee: ................................ LISSWARENET\Alain.Lissoir
118:  ||+-------------------------------------------------------------------------------------------
...:
437:  |+-----------------------------------------------------------------------------------------
438:  +-----------------------------------------------------------------------------------------
```

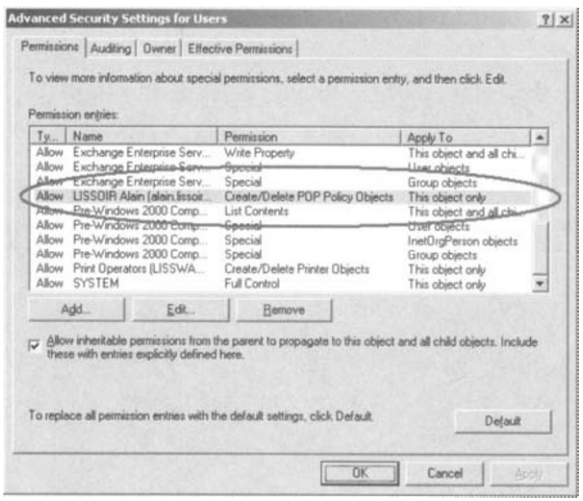


Figure 4.28 *The ACE ObjectType property used to grant or deny the creation or deletion of objects from a particular class.*

From line 105 through 118, the right "Create/Delete POP Policy Objects" is granted to trustee "Alain.Lissoir" as:

- The *ACE Type* has a value equal to ADS_ACETYPE_ACCESS_ALLOWED_OBJECT (line 113).
- The *ACE AccessMask* has a value equal to ADS_RIGHT_DS_CREATE_CHILD + ADS_RIGHT_DS_DELETE_CHILD (lines 107 and 108).
- The *ACE ObjectType* property has a GUID number corresponding to the **schemaIDGUID** of a **classSchema** object, which is the **msExchProtocolCfgPOPPolicy** object class (line 116). Check Table 4.18 to find the **schemaIDGUID** GUID number and determine the corresponding Active Directory class.

Because *ACE ObjectType* deciphering logic could be confusing, Table 4.19 summarizes the logic to follow to decipher this property.

The table must be read from left to right, column by column. For instance, it should be read as follows:

When the *ACE Type* property contains one of the values listed below ... (column 1)

```
ADS_ACETYPE_ACCESS_ALLOWED_OBJECT

ADS_ACETYPE_ACCESS_DENIED_OBJECT

ADS_ACETYPE_SYSTEM_AUDIT_OBJECT
```

... and if the "ACE AccessMask" property contains ... (column 2)

```
ADS_RIGHT_DS_READ_PROP  (0x10)
```

... then is that an Extended Right? (column 3)

Yes!

In such a case, the GUID number in the *ACE ObjectType* value refers ... (column 4)

*the GUID number from the **rightsGUID** attribute of the **controlAccessRight** object,*

*... which contains a **validAccesses** value of... (column 5)*

```
ADS_RIGHT_DS_READ_PROP Or ADS_RIGHT_DS_WRITE_PROP (0x30)
```

Table 4.18 *The schemaIDGUID GUID Number with iTs Class Names*

Display Name	Object Class	schemaIDGUID
* objects	msExchDomainContentConfig	{ab3a1ad1-1df5-11d3-aa5e-00c04f8eedd8}
account objects	account	{2628a46a-a6ad-4ae0-b854-2b12d9fe6f9e}
aCSPolicy objects	aCSPolicy	{7f561288-5301-11d1-a9c5-0000f80367c1}
aCSResourceLimits objects	aCSResourceLimits	{2e899b04-2834-11d3-91d4-0000f87a57d4}
aCSSubnet objects	aCSSubnet	{7f561289-5301-11d1-a9c5-0000f80367c1}
Active Directory Service objects	nTDSService	{19195a5f-6da0-11d0-afd3-00c04fd930c9}
ADC Connection Agreement objects	msExchConnectionAgreement	{ee64c93a-a980-11d2-a9ff-00c04f8eedd8}
ADC Schema Map Policy objects	msExchSchemaMapPolicy	{348af8f2-a982-11d2-a9ff-00c04f8eedd8}
ADC Service objects	msExchActiveDirectoryConnector	{e605672c-a980-11d2-a9ff-00c04f8eedd8}
Address List objects	addressBookContainer	{3e74f60f-3e73-11d1-a9c0-0000f80367c1}
Address Template objects	addressTemplate	{5fd4250a-1262-11d0-a060-00aa006c33ed}
Address Type objects	addrType	{a8df74ab-c5ea-11d1-bbcb-0080c76670c0}
Addressing Policy objects	msExchAddressingPolicy	{e7211f02-a980-11d2-a9ff-00c04f8eedd8}
Administrative Group objects	msExchAdminGroup	{e768a58e-a980-11d2-a9ff-00c04f8eedd8}
Administrative Groups objects	msExchAdminGroupContainer	{e7a44058-a980-11d2-a9ff-00c04f8eedd8}
Administrative Role objects	msExchAdminRole	{e7f2edf2-a980-11d2-a9ff-00c04f8eedd8}
Advanced Security objects	msExchAdvancedSecurityContainer	{8cc8fb0e-b09e-11d2-aa06-00c04f8eedd8}
applicationEntity objects	applicationEntity	{3fdfee4f-47f4-11d1-a9c3-0000f80367c1}
applicationProcess objects	applicationProcess	{5fd4250b-1262-11d0-a060-00aa006c33ed}
applicationSettings objects	applicationSettings	{f780acc1-56f0-11d1-a9c6-0000f80367c1}
applicationSiteSettings objects	applicationSiteSettings	{19195a5c-6da0-11d0-afd3-00c04fd930c9}
applicationVersion objects	applicationVersion	{ddc790ac-af4d-442a-8f0f-a1d4caa7dd92}
builtinDomain objects	builtinDomain	{bf967a81-0de6-11d0-a285-00aa003049e2}
categoryRegistration objects	categoryRegistration	{7d6c0e9d-7e20-11d0-afd6-00c04fd930c9}
cc:Mail Connector objects	msExchccMailConnector	{e85710b6-a980-11d2-a9ff-00c04f8eedd8}
Certificate Template objects	pKICertificateTemplate	{e5209ca2-3bba-11d2-90cc-00c04fd91ab1}
Certification Authority objects	certificationAuthority	{3fdfee50-47f4-11d1-a9c3-0000f80367c1}
Chat Network objects	msExchChatNetwork	{e934cb68-a980-11d2-a9ff-00c04f8eedd8}
Chat Protocol objects	msExchChatProtocol	{e9621816-a980-11d2-a9ff-00c04f8eedd8}
classRegistration objects	classRegistration	{bf967a82-0de6-11d0-a285-00aa003049e2}
classStore objects	classStore	{bf967a84-0de6-11d0-a285-00aa003049e2}
comConnectionPoint objects	comConnectionPoint	{bf967a85-0de6-11d0-a285-00aa003049e2}
Computer objects	computer	{bf967a86-0de6-11d0-a285-00aa003049e2}
Computer Policy objects	msExchComputerPolicy	{ed2c752c-a980-11d2-a9ff-00c04f8eedd8}
Conference Site objects	msExchConferenceSite	{eddce330-a980-11d2-a9ff-00c04f8eedd8}
Conference Sites objects	msExchConferenceContainer	{ed7fe77a-a980-11d2-a9ff-00c04f8eedd8}
configuration objects	configuration	{bf967a87-0de6-11d0-a285-00aa003049e2}
Connection objects	nTDSConnection	{19195a60-6da0-11d0-afd3-00c04fd930c9}
connectionPoint objects	connectionPoint	{5cb41ecf-0e4c-11d0-a286-00aa003049e2}
Connections objects	msExchConnectors	{eee325dc-a980-11d2-a9ff-00c04f8eedd8}
Contact objects	contact	{5cb41ed0-0e4c-11d0-a286-00aa003049e2}
Container objects	container	{bf967a8b-0de6-11d0-a285-00aa003049e2}
country objects	country	{bf967a8c-0de6-11d0-a285-00aa003049e2}
cRLDistributionPoint objects	cRLDistributionPoint	{167758ca-47f3-11d1-a9c3-0000f80367c1}
crossRef objects	crossRef	{bf967a8d-0de6-11d0-a285-00aa003049e2}
crossRefContainer objects	crossRefContainer	{ef9e60e0-56f7-11d1-a9c6-0000f80367c1}
Data Conference Server (T.120 MCU) objects	msExchMCU	{038680ec-a981-11d2-a9ff-00c04f8eedd8}
Data Conference Technology Provider (T.120 MCU) objects	msExchMCUContainer	{03aa4432-a981-11d2-a9ff-00c04f8eedd8}
device objects	device	{bf967a8e-0de6-11d0-a285-00aa003049e2}
dfsConfiguration objects	dfsConfiguration	{8447f9f2-1027-11d0-a05f-00aa006c33ed}
dHCPClass objects	dHCPClass	{963d2756-48be-11d1-a9c3-0000f80367c1}
Directory objects	dSA	{3fdfee52-47f4-11d1-a9c3-0000f80367c1}
Directory Replication Connector objects	msExchReplicationConnector	{99f58682-12e8-11d3-aa58-00c04f8eedd8}
Directory Synchronization objects	localDXA	{a8df74b5-c5ea-11d1-bbcb-0080c76670c0}
Directory Synchronization Requestor objects	dXRequestor	{a8df74ae-c5ea-11d1-bbcb-0080c76670c0}
Directory Synchronization Server Connector objects	dXServerConn	{a8df74af-c5ea-11d1-bbcb-0080c76670c0}
Directory Synchronization Site Server objects	dXASiteServer	{a8df74b0-c5ea-11d1-bbcb-0080c76670c0}
Display Template objects	displayTemplate	{5fd4250c-1262-11d0-a060-00aa006c33ed}
displaySpecifier objects	displaySpecifier	{e0fa1e8a-9b45-11d0-afdd-00c04fd930c9}
dnsNode objects	dnsNode	{e0fa1e8c-9b45-11d0-afdd-00c04fd930c9}
dnsZone objects	dnsZone	{e0fa1e8b-9b45-11d0-afdd-00c04fd930c9}
document objects	document	{39bad96d-c2d6-4baf-88ab-7e4207600117}
documentSeries objects	documentSeries	{7a2be07c-302f-4b96-bc90-0795d66885f8}
Domain Controller Settings objects	nTDSDSA	{f0f8ffab-1191-11d0-a060-00aa006c33ed}
domain objects	domain	{19195a5a-6da0-11d0-afd3-00c04fd930c9}
Domain objects	domainDNS	{19195a5b-6da0-11d0-afd3-00c04fd930c9}
Domain Policy objects	domainPolicy	{bf967a99-0de6-11d0-a285-00aa003049e2}
domainRelatedObject objects	domainRelatedObject	{8bfd2d3d-efda-4549-852c-f85e137aedc6}
dSUISettings objects	dSUISettings	{09b10f14-6f93-11d2-9905-0000f87a57d4}

Table 4.18 *The schemaIDGUID GUID Number with iTs Class Names (continued)*

Display Name	Object Class	schemaIDGUID
Dynamic RAS Connector objects	rASX400Link	{a8df74d4-c5ea-11d1-bbcb-0080c76670c0}
dynamicObject objects	dynamicObject	{66d51249-3355-4c1f-b24e-81f252aca23b}
Encryption Configuration objects	encryptionCfg	{a8df74b1-c5ea-11d1-bbcb-0080c76670c0}
Exchange Add-In objects	addIn	{a8df74aa-c5ea-11d1-bbcb-0080c76670c0}
Exchange Admin Extension objects	adminExtension	{a8df74ac-c5ea-11d1-bbcb-0080c76670c0}
Exchange Configuration Container objects	msExchConfigurationContainer	{d03d6858-06f4-11d2-aa53-00c04fd7d83a}
Exchange Container objects	msExchContainer	{006c91da-a981-11d2-a9ff-00c04f8eedd8}
Exchange Organization objects	msExchOrganizationContainer	{366a319c-a982-11d2-a9ff-00c04f8eedd8}
Exchange Policies objects	msExchPoliciesContainer	{3630f92c-a982-11d2-a9ff-00c04f8eedd8}
Exchange Protocols objects	msExchProtocolCfgProtocolContainer	{90f2b634-b09e-11d2-aa06-00c04f8eedd8}
Exchange Server objects	msExchExchangeServer	{01a9aa9c-a981-11d2-a9ff-00c04f8eedd8}
Exchange Server Policy objects	msExchExchangeServerPolicy	{e497942f-1d42-11d3-aa5e-00c04f8eedd8}
Exchange Servers objects	msExchServersContainer	{346e5cba-a982-11d2-a9ff-00c04f8eedd8}
Extended Right objects	controlAccessRight	{8297931e-86d3-11d0-afda-00c04fd930c9}
fileLinkTracking objects	fileLinkTracking	{dd712229-10e4-11d0-a05f-00aa006c33ed}
fileLinkTrackingEntry objects	fileLinkTrackingEntry	{8e4eb2ed-4712-11d0-a1a0-00c04fd930c9}
Foreign Security Principal objects	foreignSecurityPrincipal	{89e31c12-8530-11d0-afda-00c04fd930c9}
friendlyCountry objects	friendlyCountry	{c498f152-dc6b-474a-9f52-7cdba3d7d351}
FRS Member objects	nTFRSMember	{2a132586-9373-11d1-aebc-0000f80367c1}
FRS Replica Set objects	nTFRSReplicaSet	{5245803a-ca6a-11d0-afff-0000f80367c1}
FRS Settings objects	nTFRSSettings	{f780acc2-56f0-11d1-a9c6-0000f80367c1}
FRS Subscriber objects	nTFRSSubscriber	{2a132588-9373-11d1-aebc-0000f80367c1}
FRS Subscriptions objects	nTFRSSubscriptions	{2a132587-9373-11d1-aebc-0000f80367c1}
fTDfs objects	fTDfs	{8447f9f3-1027-11d0-a05f-00aa006c33ed}
Gateway objects	mailGateway	{a8df74b7-c5ea-11d1-bbcb-0080c76670c0}
Group objects	group	{bf967a9c-0de6-11d0-a285-00aa003049e2}
groupOfNames objects	groupOfNames	{bf967a9d-0de6-11d0-a285-00aa003049e2}
groupOfUniqueNames objects	groupOfUniqueNames	{0310a911-93a3-4e21-a7a3-55d85ab2c48b}
groupPolicyContainer objects	groupPolicyContainer	{f30e3bc2-9ff0-11d1-b603-0000f80367c1}
GroupWise Connector objects	msExchGroupWiseConnector	{91eaaac4-b09e-11d2-aa06-00c04f8eedd8}
HTTP Protocol objects	msExchProtocolCfgHTTPContainer	{9432cae6-b09e-11d2-aa06-00c04f8eedd8}
HTTP Virtual Directory objects	msExchProtocolCfgHTTPVirtualDirectory	{8c3c5050-b09e-11d2-aa06-00c04f8eedd8}
HTTP Virtual Server objects	protocolCfgHTTPServer	{a8df74c2-c5ea-11d1-bbcb-0080c76670c0}
IMAP Policy objects	msExchProtocolCfgIMAPPolicy	{35f7c0bc-a982-11d2-a9ff-00c04f8eedd8}
IMAP Protocol objects	msExchProtocolCfgIMAPContainer	{93da93e4-b09e-11d2-aa06-00c04f8eedd8}
IMAP Sessions objects	msExchProtocolCfgIMAPSessions	{99f58672-12e8-11d3-aa58-00c04f8eedd8}
IMAP Virtual Server objects	protocolCfgIMAPServer	{a8df74c5-c5ea-11d1-bbcb-0080c76670c0}
indexServerCatalog objects	indexServerCatalog	{7bfdcb8a-4807-11d1-a9c3-0000f80367c1}
inetOrgPerson objects	inetOrgPerson	{4828cc14-1437-45bc-9b07-ad6f015e5f28}
Information Store objects	msExchInformationStore	{031b371a-a981-11d2-a9ff-00c04f8eedd8}
infrastructureUpdate objects	infrastructureUpdate	{2df90d89-009f-11d2-aa4c-00c04fd7d83a}
Instant Messaging Global Settings objects	msExchIMGlobalSettingsContainer	{9f116eb8-284e-11d3-aa68-00c04f8eedd8}
Instant Messaging Protocol objects	msExchProtocolCfgIMContainer	{9f116ea3-284e-11d3-aa68-00c04f8eedd8}
Instant Messaging Virtual Server objects	msExchProtocolCfgIMVirtualServer	{9f116eb4-284e-11d3-aa68-00c04f8eedd8}
IntelliMirror Group objects	intellimirrorGroup	{07383086-91df-11d1-aebc-0000f80367c1}
IntelliMirror Service objects	intellimirrorSCP	{07383085-91df-11d1-aebc-0000f80367c1}
Internet Message Formats objects	msExchContentConfigContainer	{ab3a1acc-1df5-11d3-aa5e-00c04f8eedd8}
Inter-Site Transport objects	interSiteTransport	{26d97376-6070-11d1-a9c6-0000f80367c1}
Inter-Site Transports Container objects	interSiteTransportContainer	{26d97375-6070-11d1-a9c6-0000f80367c1}
ipsecBase objects	ipsecBase	{b40ff825-427a-11d1-a9c2-0000f80367c1}
ipsecFilter objects	ipsecFilter	{b40ff826-427a-11d1-a9c2-0000f80367c1}
ipsecISAKMPPolicy objects	ipsecISAKMPPolicy	{b40ff828-427a-11d1-a9c2-0000f80367c1}
ipsecNegotiationPolicy objects	ipsecNegotiationPolicy	{b40ff827-427a-11d1-a9c2-0000f80367c1}
ipsecNFA objects	ipsecNFA	{b40ff829-427a-11d1-a9c2-0000f80367c1}
ipsecPolicy objects	ipsecPolicy	{b7b13121-b82e-11d0-afee-0000f80367c1}
Key Management Server objects	msExchKeyManagementServer	{8ce334ec-b09e-11d2-aa06-00c04f8eedd8}
leaf objects	leaf	{bf967a9e-0de6-11d0-a285-00aa003049e2}
Licensing Site Settings objects	licensingSiteSettings	{1be8f17d-a9ff-11d0-afe2-00c04fd930c9}
linkTrackObjectMoveTable objects	linkTrackObjectMoveTable	{ddac0cf5-af8f-11d0-afeb-00c04fd930c9}
linkTrackOMTEntry objects	linkTrackOMTEntry	{ddac0cf7-af8f-11d0-afeb-00c04fd930c9}
linkTrackVolEntry objects	linkTrackVolEntry	{ddac0cf6-af8f-11d0-afeb-00c04fd930c9}
linkTrackVolumeTable objects	linkTrackVolumeTable	{ddac0cf4-af8f-11d0-afeb-00c04fd930c9}
locality objects	locality	{bf967aa0-0de6-11d0-a285-00aa003049e2}
lostAndFound objects	lostAndFound	{52ab8671-5709-11d1-a9c6-0000f80367c1}
Mail Recipient objects	mailRecipient	{bf967aa1-0de6-11d0-a285-00aa003049e2}
meeting objects	meeting	{11b6cc94-48c4-11d1-a9c3-0000f80367c1}
Message Delivery Configuration objects	msExchMessageDeliveryConfig	{ab3a1ad7-1df5-11d3-aa5e-00c04f8eedd8}
Message Gateway for cc:Mail objects	mailConnector	{a8df74b6-c5ea-11d1-bbcb-0080c76670c0}
Message Transfer Agent objects	mTA	{a8df74a7-c5ea-11d1-bbcb-0080c76670c0}

Table 4.18 *The schemaIDGUID GUID Number with iTs Class Names (continued)*

Display Name	Object Class	schemaIDGUID
mHSMonitoringConfig objects	mHSMonitoringConfig	{a8df74bb-c5ea-11d1-bbcb-0080c76670c0}
Microsoft Exchange System Objects objects	msExchSystemObjectsContainer	{0bffa04c-7d8e-44cd-968a-b2cac11d17e1}
Monitoring Link Configuration objects	mHSLinkMonitoringConfig	{a8df74b9-c5ea-11d1-bbcb-0080c76670c0}
Monitoring Server Configuration objects	mHSServerMonitoringConfig	{a8df74bd-c5ea-11d1-bbcb-0080c76670c0}
msCOM-Partition objects	msCOM-Partition	{c9010e74-4e58-49f7-8a89-5e3e2340fcf8}
msCOM-PartitionSet objects	msCOM-PartitionSet	{250464ab-c417-497a-975a-9e0d459a7ca1}
msDS-App-Configuration objects	msDS-App-Configuration	{90df3c3e-1854-4455-a5d7-cad40d56657a}
msDS-AppData objects	msDS-AppData	{9e67d761-e327-4d55-bc95-682f875e2f8e}
msDS-AzAdminManager objects	msDS-AzAdminManager	{cfee1051-5f28-4bae-a863-5d0cc18a8ed1}
msDS-AzApplication objects	msDS-AzApplication	{ddf8de9b-cba5-4e12-842e-28d8b66f75ec}
msDS-AzOperation objects	msDS-AzOperation	{860abe37-9a9b-4fa4-b3d2-b8ace5df9ec5}
msDS-AzRole objects	msDS-AzRole	{8213eac9-9d55-44dc-925c-e9a52b927644}
msDS-AzScope objects	msDS-AzScope	{4feae054-ce55-47bb-860e-5b12063a51de}
msDS-AzTask objects	msDS-AzTask	{1ed3a473-9b1b-418a-bfa0-3a37b95a5306}
msExchAddressListServiceContainer objects	msExchAddressListServiceContainer	{b1fce95a-1d44-11d3-aa5e-00c04f8eedd8}
msExchBaseClass objects	msExchBaseClass	{d8782c34-46ca-11d3-aa72-00c04f8eedd8}
msExchCalendarConnector objects	msExchCalendarConnector	{922180da-b09e-11d2-aa06-00c04f8eedd8}
msExchCertificateInformation objects	msExchCertificateInformation	{e8977034-a980-11d2-a9ff-00c04f8eedd8}
msExchChatBan objects	msExchChatBan	{e8d0a8a4-a980-11d2-a9ff-00c04f8eedd8}
msExchChatChannel objects	msExchChatChannel	{e902ba06-a980-11d2-a9ff-00c04f8eedd8}
msExchChatUserClass objects	msExchChatUserClass	{e9a0153a-a980-11d2-a9ff-00c04f8eedd8}
msExchConnector objects	msExchConnector	{89652316-b09e-11d2-aa06-00c04f8eedd8}
msExchCTP objects	msExchCTP	{00aa8efe-a981-11d2-a9ff-00c04f8eedd8}
msExchCustomAttributes objects	msExchCustomAttributes	{00e629c8-a981-11d2-a9ff-00c04f8eedd8}
msExchDynamicDistributionList objects	msExchDynamicDistributionList	{018849b0-a981-11d2-a9ff-00c04f8eedd8}
msExchGenericPolicy objects	msExchGenericPolicy	{e32977cd-1d31-11d3-aa5e-00c04f8eedd8}
msExchGenericPolicyContainer objects	msExchGenericPolicyContainer	{e32977c3-1d31-11d3-aa5e-00c04f8eedd8}
msExchIMFirewall objects	msExchIMFirewall	{9f116ebe-284e-11d3-aa68-00c04f8eedd8}
msExchIMRecipient objects	msExchIMRecipient	{028502f4-a981-11d2-a9ff-00c04f8eedd8}
msExchMailboxManagerPolicy objects	msExchMailboxManagerPolicy	{36f94fcc-ebbb-4a32-b721-1cae42b2dbab}
msExchMailStorage objects	msExchMailStorage	{03652000-a981-11d2-a9ff-00c04f8eedd8}
msExchMDB objects	msExchMDB	{03d069d2-a981-11d2-a9ff-00c04f8eedd8}
msExchMonitorsContainer objects	msExchMonitorsContainer	{03f68f72-a981-11d2-a9ff-00c04f8eedd8}
msExchMultiMediaUser objects	msExchMultiMediaUser	{1529cf7a-2fdb-11d3-aa6d-00c04f8eedd8}
msExchOVVMConnector objects	msExchOVVMConnector	{91ce0e8c-b09e-11d2-aa06-00c04f8eedd8}
msExchPrivateMDBProxy objects	msExchPrivateMDBProxy	{b8d47e54-4b78-11d3-aa75-00c04f8eedd8}
msExchProtocolCfgHTTPFilter objects	msExchProtocolCfgHTTPFilter	{8c7588c0-b09e-11d2-aa06-00c04f8eedd8}
msExchProtocolCfgHTTPFilters objects	msExchProtocolCfgHTTPFilters	{8c58ec88-b09e-11d2-aa06-00c04f8eedd8}
msExchProtocolCfgIM objects	msExchProtocolCfgIM	{9f116ea7-284e-11d3-aa68-00c04f8eedd8}
msExchProtocolCfgSharedContainer objects	msExchProtocolCfgSharedContainer	{939ef91a-b09e-11d2-aa06-00c04f8eedd8}
msExchProtocolCfgSMTPIPAddress objects	msExchProtocolCfgSMTPIPAddress	{8b7b31d6-b09e-11d2-aa06-00c04f8eedd8}
msExchProtocolCfgSMTPIPAddressContainer objects	msExchProtocolCfgSMTPIPAddressContainer	{8b2c843c-b09e-11d2-aa06-00c04f8eedd8}
msExchPseudoPF objects	msExchPseudoPF	{cec4472b-22ae-11d3-aa62-00c04f8eedd8}
msExchPseudoPFAdmin objects	msExchPseudoPFAdmin	{9ae2fa1b-22b0-11d3-aa62-00c04f8eedd8}
msExchPublicFolderTreeContainer objects	msExchPublicFolderTreeContainer	{3582ed82-a982-11d2-a9ff-00c04f8eedd8}
msExchSNADSConnector objects	msExchSNADSConnector	{91b17254-b09e-11d2-aa06-00c04f8eedd8}
msieee80211-Policy objects	msieee80211-Policy	{7b9a2d92-b7eb-4382-9772-c3e0f9baaf94}
MSMail Connector objects	mSMailConnector	{a8df74be-c5ea-11d1-bbcb-0080c76670c0}
MSMQ Configuration objects	mSMQConfiguration	{9a0dc344-c100-11d1-bbc5-0080c76670c0}
MSMQ Enterprise objects	mSMQEnterpriseSettings	{9a0dc345-c100-11d1-bbc5-0080c76670c0}
MSMQ Group objects	msMQ-Group	{46b27aac-aafa-4ffb-b773-e5bf621ee87b}
MSMQ Queue Alias objects	msMQ-Custom-Recipient	{876d6817-35cc-436c-acea-5ef7174dd9be}
MSMQ Queue objects	mSMQQueue	{9a0dc343-c100-11d1-bbc5-0080c76670c0}
MSMQ Routing Link objects	mSMQSiteLink	{9a0dc346-c100-11d1-bbc5-0080c76670c0}
MSMQ Settings objects	mSMQSettings	{9a0dc347-c100-11d1-bbc5-0080c76670c0}
MSMQ Upgraded User objects	mSMQMigratedUser	{50776997-3c3d-11d2-90cc-00c04fd91ab1}
msPKI-Enterprise-Oid objects	msPKI-Enterprise-Oid	{37cfd85c-6719-4ad8-8f9e-8678ba627563}
msPKI-Key-Recovery-Agent objects	msPKI-Key-Recovery-Agent	{26ccf238-a08e-4b86-9a82-a8c9ac7ee5cb}
msPKI-PrivateKeyRecoveryAgent objects	msPKI-PrivateKeyRecoveryAgent	{1562a632-44b9-4a7e-a2d3-e426c96a3acc}
mS-SQL-OLAPCube objects	mS-SQL-OLAPCube	{09f0506a-cd28-11d2-9993-0000f87a57d4}
mS-SQL-OLAPDatabase objects	mS-SQL-OLAPDatabase	{20af031a-ccef-11d2-9993-0000f87a57d4}
mS-SQL-OLAPServer objects	mS-SQL-OLAPServer	{0c7e18ea-ccef-11d2-9993-0000f87a57d4}
mS-SQL-SQLDatabase objects	mS-SQL-SQLDatabase	{1d08694a-ccef-11d2-9993-0000f87a57d4}
mS-SQL-SQLPublication objects	mS-SQL-SQLPublication	{17c2f64e-ccef-11d2-9993-0000f87a57d4}
mS-SQL-SQLRepository objects	mS-SQL-SQLRepository	{11d43c5c-ccef-11d2-9993-0000f87a57d4}
mS-SQL-SQLServer objects	mS-SQL-SQLServer	{05f6c878-ccef-11d2-9993-0000f87a57d4}
msTAPI-RtConference objects	msTAPI-RtConference	{ca7b9735-4b2a-4e49-89c3-99025334dc94}
msTAPI-RtPerson objects	msTAPI-RtPerson	{53ea1cb5-b704-4df9-818f-5cb4ec86cac1}
msWMI-IntRangeParam objects	msWMI-IntRangeParam	{50ca5d7d-5c8b-4ef3-b9df-5b66d491e526}

Table 4.18 *The schemaIDGUID GUID Number with iTs Class Names (continued)*

Display Name	Object Class	schemaIDGUID
msWMI-IntSetParam objects	msWMI-IntSetParam	{292f0d9a-cf76-42b0-841f-b650f331df62}
msWMI-MergeablePolicyTemplate objects	msWMI-MergeablePolicyTemplate	{07502414-fdca-4851-b04a-13645b11d226}
msWMI-ObjectEncoding objects	msWMI-ObjectEncoding	{55dd81c9-c312-41f9-a84d-c6adbdf1e8e1}
msWMI-PolicyTemplate objects	msWMI-PolicyTemplate	{e2bc80f1-244a-4d59-acc6-ca5c4f82e6e1}
msWMI-PolicyType objects	msWMI-PolicyType	{595b2613-4109-4e77-9013-a3bb4ef277c7}
msWMI-RangeParam objects	msWMI-RangeParam	{45fb5a57-5018-4d0f-9056-997c8c9122d9}
msWMI-RealRangeParam objects	msWMI-RealRangeParam	{6afe8fe2-70bc-4cce-b166-a96f7359c514}
msWMI-Rule objects	msWMI-Rule	{3c7e6f83-dd0e-481b-a0c2-74cd96ef2a66}
msWMI-ShadowObject objects	msWMI-ShadowObject	{f1e44bdf-8dd3-4235-9c86-f91f31f5b569}
msWMI-SimplePolicyTemplate objects	msWMI-SimplePolicyTemplate	{6cc8b2b5-12df-44f6-8307-e74f5cdee369}
msWMI-Som objects	msWMI-Som	{ab857078-0142-4406-945b-34c9b6b13372}
msWMI-StringSetParam objects	msWMI-StringSetParam	{0bc579a2-1da7-4cea-b699-807f3b9d63a4}
msWMI-UintRangeParam objects	msWMI-UintRangeParam	{d9a799b2-cef3-48b3-b5ad-fb85f8dd3214}
msWMI-UintSetParam objects	msWMI-UintSetParam	{8f4beb31-4e19-46f5-932e-5fa03c339b1d}
msWMI-UnknownRangeParam objects	msWMI-UnknownRangeParam	{b82ac26b-c6db-4098-92c6-49c18a3336e1}
msWMI-WMIGPO objects	msWMI-WMIGPO	{05630000-3927-4ede-bf27-ca91f275c26f}
NNTP Protocol objects	msExchProtocolCfgNNTPContainer	{94162eae-b09e-11d2-aa06-00c04f8eedd8}
NNTP Virtual Server objects	protocolCfgNNTPServer	{a8df74cb-c5ea-11d1-bbcb-0080c76670c0}
Notes Connector objects	msExchNotesConnector	{04c85e62-a981-11d2-a9ff-00c04f8eedd8}
Offline Address List objects	msExchOAB	{3686cdd4-a982-11d2-a9ff-00c04f8eedd8}
organization objects	organization	{bf967aa3-0de6-11d0-a285-00aa003049e2}
Organizational Unit objects	organizationalUnit	{bf967aa5-0de6-11d0-a285-00aa003049e2}
organizationalPerson objects	organizationalPerson	{bf967aa4-0de6-11d0-a285-00aa003049e2}
organizationalRole objects	organizationalRole	{a8df74bf-c5ea-11d1-bbcb-0080c76670c0}
packageRegistration objects	packageRegistration	{bf967aa6-0de6-11d0-a285-00aa003049e2}
person objects	person	{bf967aa7-0de6-11d0-a285-00aa003049e2}
physicalLocation objects	physicalLocation	{b7b13122-b82e-11d0-afee-0000f80367c1}
pKIEnrollmentService objects	pKIEnrollmentService	{ee4aa692-3bba-11d2-90cc-00c04fd91ab1}
POP Policy objects	msExchProtocolCfgPOPPolicy	{35be884c-a982-11d2-a9ff-00c04f8eedd8}
POP Protocol objects	msExchProtocolCfgPOPContainer	{93f99276-b09e-11d2-aa06-00c04f8eedd8}
POP Sessions objects	msExchProtocolCfgPOPSessions	{99f58676-12e8-11d3-aa58-00c04f8eedd8}
POP Virtual Server objects	protocolCfgPOPServer	{a8df74ce-c5ea-11d1-bbcb-0080c76670c0}
Printer objects	printQueue	{bf967aa8-0de6-11d0-a285-00aa003049e2}
Private Information Store objects	msExchPrivateMDB	{36145cf4-a982-11d2-a9ff-00c04f8eedd8}
Private Information Store Policy objects	msExchPrivateMDBPolicy	{35db2484-a982-11d2-a9ff-00c04f8eedd8}
protocolCfg objects	protocolCfg	{a8df74c0-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgHTTP objects	protocolCfgHTTP	{a8df74c1-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgIMAP objects	protocolCfgIMAP	{a8df74c4-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgLDAP objects	protocolCfgLDAP	{a8df74c7-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgNNTP objects	protocolCfgNNTP	{a8df74ca-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgPOP objects	protocolCfgPOP	{a8df74cd-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgShared objects	protocolCfgShared	{a8df74d0-c5ea-11d1-bbcb-0080c76670c0}
protocolCfgSMTP objects	protocolCfgSMTP	{33f98980-a982-11d2-a9ff-00c04f8eedd8}
Public Folder objects	publicFolder	{f0f8ffac-1191-11d0-a060-00aa006c33ed}
Public Folder Top Level Hierarchy objects	msExchPFTree	{364d9564-a982-11d2-a9ff-00c04f8eedd8}
Public Information Store objects	msExchPublicMDB	{3568b3a4-a982-11d2-a9ff-00c04f8eedd8}
Public Information Store Policy objects	msExchPublicMDBPolicy	{354c176c-a982-11d2-a9ff-00c04f8eedd8}
Query Policy objects	queryPolicy	{83cc7075-cca7-11d0-afff-0000f80367c1}
RAS MTA Transport Stack objects	rASStack	{a8df74d3-c5ea-11d1-bbcb-0080c76670c0}
Recipient Policies objects	msExchRecipientPolicyContainer	{e32977d2-1d31-11d3-aa5e-00c04f8eedd8}
Recipient Policy objects	msExchRecipientPolicy	{e32977d8-1d31-11d3-aa5e-00c04f8eedd8}
Recipient Update Service objects	msExchAddressListService	{e6a2c260-a980-11d2-a9ff-00c04f8eedd8}
Remote Storage Service objects	remoteStorageServicePoint	{2a39c5bd-8960-11d1-aebc-0000f80367c1}
remoteDXA objects	remoteDXA	{a8df74d5-c5ea-11d1-bbcb-0080c76670c0}
remoteMailRecipient objects	remoteMailRecipient	{bf967aa9-0de6-11d0-a285-00aa003049e2}
Replication Connectors objects	msExchReplicationConnectorContainer	{99f5867e-12e8-11d3-aa58-00c04f8eedd8}
residentialPerson objects	residentialPerson	{a8df74d6-c5ea-11d1-bbcb-0080c76670c0}
rFC822LocalPart objects	rFC822LocalPart	{b93e3a78-cbae-485e-a07b-5ef4ae505686}
rIDManager objects	rIDManager	{6617188d-8f3c-11d0-afda-00c04fd930c9}
rIDSet objects	rIDSet	{7bfdcb89-4807-11d1-a9c3-0000f80367c1}
room objects	room	{7860e5d2-c8b0-4cbb-bd45-d9455beb9206}
Routing Group Connector objects	msExchRoutingGroupConnector	{899e5b86-b09e-11d2-aa06-00c04f8eedd8}
Routing Group objects	msExchRoutingGroup	{35154156-a982-11d2-a9ff-00c04f8eedd8}
Routing Groups objects	msExchRoutingGroupContainer	{34de6b40-a982-11d2-a9ff-00c04f8eedd8}
RPC Services objects	rpcContainer	{80212842-4bdc-11d1-a9c4-0000f80367c1}
rpcEntry objects	rpcEntry	{bf967aac-0de6-11d0-a285-00aa003049e2}
rpcGroup objects	rpcGroup	{88611bdf-8cf4-11d0-afda-00c04fd930c9}
rpcProfile objects	rpcProfile	{88611be1-8cf4-11d0-afda-00c04fd930c9}
rpcProfileElement objects	rpcProfileElement	{f29653cf-7ad0-11d0-afd6-00c04fd930c9}

Table 4.18 *The schemaIDGUID GUID Number with iTs Class Names (continued)*

Display Name	Object Class	schemaIDGUID
rpcServer objects	rpcServer	{88611be0-8cf4-11d0-afda-00c04fd930c9}
rpcServerElement objects	rpcServerElement	{f29653d0-7ad0-11d0-afd6-00c04fd930c9}
rRASAdministrationConnectionPoint objects	rRASAdministrationConnectionPoint	{2a39c5be-8960-11d1-aebc-0000f80367c1}
rRASAdministrationDictionary objects	rRASAdministrationDictionary	{f39b98ae-938d-11d1-aebd-0000f80367c1}
samDomain objects	samDomain	{bf967a90-0de6-11d0-a285-00aa003049e2}
samDomainBase objects	samDomainBase	{bf967a91-0de6-11d0-a285-00aa003049e2}
samServer objects	samServer	{bf967aad-0de6-11d0-a285-00aa003049e2}
Schedule+ Free/Busy Connector objects	msExchSchedulePlusConnector	{b1fce946-1d44-11d3-aa5e-00c04f8eedd8}
Schema Attribute objects	attributeSchema	{bf967a80-0de6-11d0-a285-00aa003049e2}
Schema Container objects	dMD	{bf967a8f-0de6-11d0-a285-00aa003049e2}
Schema Object objects	classSchema	{bf967a83-0de6-11d0-a285-00aa003049e2}
secret objects	secret	{bf967aae-0de6-11d0-a285-00aa003049e2}
securityObject objects	securityObject	{bf967aaf-0de6-11d0-a285-00aa003049e2}
securityPrincipal objects	securityPrincipal	{bf967ab0-0de6-11d0-a285-00aa003049e2}
Server LDAP Protocol objects	protocolCfgLDAPServer	{a8df74c8-c5ea-11d1-bbcb-0080c76670c0}
Server objects	server	{bf967a92-0de6-11d0-a285-00aa003049e2}
Server Protocols objects	protocolCfgSharedServer	{a8df74d1-c5ea-11d1-bbcb-0080c76670c0}
Servers Container objects	serversContainer	{f780acc0-56f0-11d1-a9c6-0000f80367c1}
Service objects	serviceAdministrationPoint	{b7b13123-b82e-11d0-afee-0000f80367c1}
serviceClass objects	serviceClass	{bf967ab1-0de6-11d0-a285-00aa003049e2}
serviceConnectionPoint objects	serviceConnectionPoint	{28630ec1-41d5-11d1-a9c1-0000f80367c1}
serviceInstance objects	serviceInstance	{bf967ab2-0de6-11d0-a285-00aa003049e2}
Shared Folder objects	volume	{bf967abb-0de6-11d0-a285-00aa003049e2}
simpleSecurityObject objects	simpleSecurityObject	{5fe69b0b-e146-4f15-b0ab-c1e5d488e094}
Site Addressing objects	siteAddressing	{a8df74d9-c5ea-11d1-bbcb-0080c76670c0}
Site Connector objects	siteConnector	{a8df74da-c5ea-11d1-bbcb-0080c76670c0}
Site HTTP Protocol objects	protocolCfgHTTPSite	{a8df74c3-c5ea-11d1-bbcb-0080c76670c0}
Site IMAP Protocol objects	protocolCfgIMAPSite	{a8df74c6-c5ea-11d1-bbcb-0080c76670c0}
Site LDAP Protocol objects	protocolCfgLDAPSite	{a8df74c9-c5ea-11d1-bbcb-0080c76670c0}
Site Link Bridge objects	siteLinkBridge	{d50c2cdf-8951-11d1-aebc-0000f80367c1}
Site Link objects	siteLink	{d50c2cde-8951-11d1-aebc-0000f80367c1}
Site MTA Configuration objects	mTACfg	{a8df74a8-c5ea-11d1-bbcb-0080c76670c0}
Site NNTP Protocol objects	protocolCfgNNTPSite	{a8df74cc-c5ea-11d1-bbcb-0080c76670c0}
Site objects	site	{bf967ab3-0de6-11d0-a285-00aa003049e2}
Site POP Protocol objects	protocolCfgPOPSite	{a8df74cf-c5ea-11d1-bbcb-0080c76670c0}
Site Protocols objects	protocolCfgSharedSite	{a8df74d2-c5ea-11d1-bbcb-0080c76670c0}
Site Replication Service objects	msExchSiteReplicationService	{99f5867b-12e8-11d3-aa58-00c04f8eedd8}
Site Settings objects	nTDSSiteSettings	{19195a5d-6da0-11d0-afd3-00c04fd930c9}
Site SMTP Protocol objects	protocolCfgSMTPSite	{32f0e47a-a982-11d2-a9ff-00c04f8eedd8}
Sites Container objects	sitesContainer	{7a4117da-cd67-11d0-afff-0000f80367c1}
SMTP Connector objects	msExchRoutingSMTPConnector	{89baf7be-b09e-11d2-aa06-00c04f8eedd8}
SMTP Domain objects	protocolCfgSMTPDomain	{33d82894-a982-11d2-a9ff-00c04f8eedd8}
SMTP Domains objects	protocolCfgSMTPDomainContainer	{33bb8c5c-a982-11d2-a9ff-00c04f8eedd8}
SMTP Policy objects	msExchProtocolCfgSMTPPolicy	{359f89ba-a982-11d2-a9ff-00c04f8eedd8}
SMTP Protocol objects	msExchProtocolCfgSMTPContainer	{93bb9552-b09e-11d2-aa06-00c04f8eedd8}
SMTP Routing Sources objects	protocolCfgSMTPRoutingSources	{3397c916-a982-11d2-a9ff-00c04f8eedd8}
SMTP Sessions objects	protocolCfgSMTPSessions	{8ef628c6-b093-11d2-aa06-00c04f8eedd8}
SMTP Turf List objects	msExchSMTPTurfList	{0b836da5-3b20-11d3-aa6f-00c04f8eedd8}
SMTP Virtual Server objects	protocolCfgSMTPServer	{3378ca84-a982-11d2-a9ff-00c04f8eedd8}
Storage Group objects	msExchStorageGroup	{3435244a-a982-11d2-a9ff-00c04f8eedd8}
storage objects	storage	{bf967ab5-0de6-11d0-a285-00aa003049e2}
Subnet objects	subnet	{b7b13124-b82e-11d0-afee-0000f80367c1}
Subnets Container objects	subnetContainer	{b7b13125-b82e-11d0-afee-0000f80367c1}
subSchema objects	subSchema	{5a8b3261-c38d-11d1-bbc9-0080c76670c0}
System Attendant objects	exchangeAdminService	{a8df74b2-c5ea-11d1-bbcb-0080c76670c0}
System Policies objects	msExchSystemPolicyContainer	{32412a7a-22af-479c-a444-624c0137122e}
System Policy objects	msExchSystemPolicy	{ba085a33-8807-4c6c-9522-2cf5a2a5e9c2}
TCP (RFC1006) MTA Transport Stack objects	rFC1006Stack	{a8df74d7-c5ea-11d1-bbcb-0080c76670c0}
TCP (RFC1006) X.400 Connector objects	rFC1006X400Link	{a8df74d8-c5ea-11d1-bbcb-0080c76670c0}
top objects	top	{bf967ab7-0de6-11d0-a285-00aa003049e2}
TP4 MTA Transport Stack objects	tP4Stack	{a8df74db-c5ea-11d1-bbcb-0080c76670c0}
TP4 X.400 Connector objects	tP4X400Link	{a8df74dc-c5ea-11d1-bbcb-0080c76670c0}
transportStack objects	transportStack	{a8df74dd-c5ea-11d1-bbcb-0080c76670c0}
Trusted Domain objects	trustedDomain	{bf967ab8-0de6-11d0-a285-00aa003049e2}
typeLibrary objects	typeLibrary	{281416e2-1968-11d0-a28f-00aa003049e2}
User objects	user	{bf967aba-0de6-11d0-a285-00aa003049e2}
Video Conference Technology Provider objects	msExchIpConfContainer	{99f5866d-12e8-11d3-aa58-00c04f8eedd8}
Virtual Chat Network objects	msExchChatVirtualNetwork	{ea5ed15a-a980-11d2-a9ff-00c04f8eedd8}
X.25 MTA Transport Stack objects	x25Stack	{a8df74de-c5ea-11d1-bbcb-0080c76670c0}
X.25 X.400 Connector objects	x25X400Link	{a8df74df-c5ea-11d1-bbcb-0080c76670c0}
x400Link objects	x400Link	{a8df74e0-c5ea-11d1-bbcb-0080c76670c0}

Table 4.19 *Summary of the GUID Number Origins for the ACE ObjectType Property*

When the *ACE Type* property contains one of the values listed below ...	... and if the *ACE AccessMask* property contains ...		... then is that an Extended Right?	In such a case, the GUID number in the *ACE ObjectType* value refers ...	... which contains a validAccesses value of ...	
ADS_ACETYPE_ACCESS_ALLOWED_OBJECT	ADS_RIGHT_DS_CONTROL_ACCESS	0x100	Yes	the GUID number of the **rightsGUID** attribute of the **controlAccessRight** object, (1)	ADS_RIGHT_DS_CONTROL_ACCESS	0x100
OR	ADS_RIGHT_DS_READ_PROP	0x10	Yes	the GUID number of the **rightsGUID** attribute of the **controlAccessRight** object, (1)	ADS_RIGHT_DS_READ_PROP	0x30
ADS_ACETYPE_ACCESS_DENIED_OBJECT	**AND/OR**				Or	
	ADS_RIGHT_DS_WRITE_PROP	0x20			ADS_RIGHT_DS_WRITE_PROP	
OR	ADS_RIGHT_DS_SELF	0x8	Yes	the GUID number of the **rightsGUID** attribute of the **controlAccessRight** object, (1)	ADS_RIGHT_DS_SELF	0x8
ADS_ACETYPE_SYSTEM_AUDIT_OBJECT	ADS_RIGHT_DS_CREATE_CHILD	0x1	No	the GUID number of the **schemaIDGUID** attribute of the **classSchema** object, (2)	N/A	
	AND/OR					
	ADS_RIGHT_DS_DELETE_CHILD	0x2				

(1) See table 4.17 for **rightsGUID** attribute values of the **controlAccessRight** objects.
(2) See table 4.18 for **schemaIDGUID** attribute values of the **classSchema** objects.

To customize the three Extended Rights samples shown in Figure 4.22, the following command lines must be used:

- For the "Personal Information" Extended Right:

```
C:\>WMIManageSD.Wsf /ADObject:"CN=LISSOIR Alain,CN=Users,DC=LissWare,DC=Net"
                    /Trustee:LissWareNET\Alain.Lissoir
                    /ACEType:ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
                    /ACEMask:ADS_RIGHT_DS_READ_PROP,
                             ADS_RIGHT_DS_WRITE_PROP
                    /ACEFlags:None
                    /ObjectType:{77B5B886-944A-11D1-AEBD-0000F80367C1}
                    /AddAce+ /ADSI+
```

- For the "Send As" Extended Right:

```
C:\>WMIManageSD.Wsf /ADObject:"CN=LISSOIR Alain,CN=Users,DC=LissWare,DC=Net"
                    /Trustee:LissWareNET\Alain.Lissoir
                    /ACEType:ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
                    /ACEMask:ADS_RIGHT_DS_CONTROL_ACCESS
                    /ACEFlags:None
                    /ObjectType:{AB721A54-1E2F-11D0-9819-00AA0040529B}
                    /AddAce+ /ADSI+
```

- For the "Add/Remove self as member" Extended Right:

```
C:\>WMIManageSD.Wsf /ADObject:"CN=Enterprise Admins,CN=Users,DC=LissWare,DC=Net"
                    /Trustee:LissWareNET\Alain.Lissoir
                    /ACEType:ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
                    /ACEMask:ADS_RIGHT_DS_SELF
                    /ACEFlags:ADS_ACEFLAG_CONTAINER_INHERIT_ACE
                    /ObjectType:{BF9679C0-0DE6-11D0-A285-00AA003049E2}
                    /AddAce+ /ADSI+
```

To customize the ACE inheritance shown in Figure 4.28 ("The *ACE ObjectType* property used to grant or deny the creation or deletion of objects from a particular class"), the following command line must be used:

```
C:\>WMIManageSD.Wsf /ADObject:"CN=Users,DC=LissWare,DC=Net"
                    /Trustee:LissWareNET\Alain.Lissoir
                    /ACEType:ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
                    /ACEMask:ADS_RIGHT_DS_CREATE_CHILD,
                             ADS_RIGHT_DS_DELETE_CHILD
                    /ACEFlags:ADS_ACEFLAG_CONTAINER_INHERIT_ACE
                    /ObjectType:{35BE884C-A982-11D2-A9FF-00C04F8EEDD8}
                    /AddAce+ /ADSI+
```

Since we manipulate a security descriptor coming from Active Directory, the ADSI security descriptor access method is used. The **/ADSI+** switch is specified for every command line. Note that the WMI access method can also be used for this example, since we manage the DACL of the security descriptor. However, as we have seen in section 4.4.4 ("Which access technique to use? Which security descriptor representation do we obtain?"), the

SACL access of an Active Directory security descriptor via WMI is not supported.

The command-line input is always based on the various deciphering outputs previously seen and the content of:

- Table 4.14, "The Active Directory Object *ACE AccessMask* Values—Standard View"
- Table 4.15, "The Active Directory Object *ACE AccessMask* Values—Advanced View"
- Table 4.16, "The Active Directory Objects *ACE Flags* Values"
- Table 4.17, "Extended Rights Available in Active Directory under Windows Server 2003 (Exchange 2000 Extended Rights Included)"
- Table 4.18, "The **schemaIDGUID** GUID Number with Its Class Names"
- Table 4.19, "Summary of the GUID Number Origins for the *ACE ObjectType* Property"

4.11.4.5.3.2 Understanding the ACE InheritedObjectType property As discussed when we examined the *ACE Flags* property (section 4.11.4.3), objects contained in subcontainers can inherit ACEs. However, with Active Directory, there are situations where only a specific class of object will inherit an ACE. Figure 4.29 shows an example of this configuration.

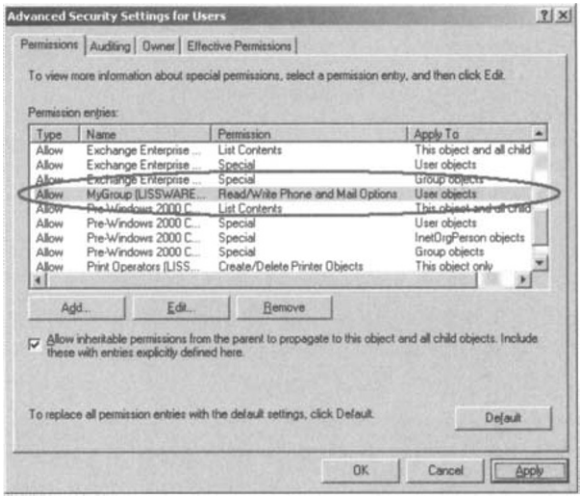


Figure 4.29
ACE Inheritance to a specific object class.

The group "MyGroup" is granted to read and write the phone and mail options on all user objects. If we decipher the ACE, we obtain the following result:

```
  1:  C:\>WMIManageSD.Wsf /ADObject:"CN=Users,DC=LissWare,DC=Net" /Decipher+ /ADSI+
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:  Reading AD object security descriptor via ADSI from 'LDAP:// CN=Users,DC=LissWareNET,...'.
  6:
  7:  +- ADSI Security Descriptor ----------------------------------------------------------------
  8:  | Owner: ............................... LISSWARENET\Domain Admins
  9:  | Group: ............................... LISSWARENET\Domain Admins
 10:  | Revision: ............................ 1
 11:  | Control: ............................. &h8C14
 ..:
 17:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
 18:  ||+- ADSI ACE ----------------------------------------------------------------------------
 ..:
 33:  ||+-------------------------------------------------------------------------------------
...:
121:  ||+- ADSI ACE ----------------------------------------------------------------------------
122:  ||| AccessMask: ........................ &h30
123:                                           ADS_RIGHT_DS_READ_PROP
124:                                           ADS_RIGHT_DS_WRITE_PROP
125:  ||| AceFlags: .......................... &hA
126:                                           ADS_ACEFLAG_CONTAINER_INHERIT_ACE
127:                                           ADS_ACEFLAG_INHERIT_ONLY_ACE
128:                                           ADS_ACEFLAG_VALID_INHERIT_FLAGS
129:  ||| AceType: ............................ &h5
130:                                           ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
131:  ||| AceFlagType: ........................ &h3
132:                                           ADS_FLAG_OBJECT_TYPE_PRESENT
133:                                           ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
134:  ||| ObjectType: ......................... {E45795B2-9455-11D1-AEBD-0000F80367C1}
135:  ||| InheritedObjectType: ................ {BF967ABA-0DE6-11D0-A285-00AA003049E2}
136:  ||| Trustee: ........................... LISSWARENET\MyGroup
137:  ||+-------------------------------------------------------------------------------------
...:
456:  |+-----------------------------------------------------------------------------------------
457:  +-----------------------------------------------------------------------------------------
```

From line 121 through 137, the "Phone and Mail options" Extended Right is granted to the trustee "MyGroup" for read and write operations as:

- The *ACE Type* has a value equal to ADS_ACETYPE_ACCESS_ALLOWED_OBJECT (line 130).
- The *ACE AccessMask* has a value equal to ADS_RIGHT_DS_READ_PROP + ADS_RIGHT_DS_WRITE_PROP (lines 123 and 124).
- The *ACE ObjectType* property has a GUID number corresponding to the "Phone and Mail options" Extended Right (line 134). Table 4.17 lists the Extended Rights GUID numbers with their corresponding display names.

- Because it applies to the user objects only, the *ACE InheritedObjectType* property is set with the GUID number stored in the **schemaIDGUID** of the **user** class (line 135). To find the name of the class with the GUID number, refer to Table 4.18.

To customize this inheritance with the script, as shown in Figure 4.29, the following command line must be used:

```
1:  C:\>WMIManageSD.Wsf /ADObject:"CN=Users,DC=LissWare,DC=Net"
2:                      /Trustee:LissWareNET\MyGroup
3:                      /ACEType:ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
4:                      /ACEMask:ADS_RIGHT_DS_READ_PROP,
5:                               ADS_RIGHT_DS_WRITE_PROP
6:                      /ACEFlags:ADS_ACEFLAG_CONTAINER_INHERIT_ACE,
7:                                ADS_ACEFLAG_INHERIT_ONLY_ACE
8:                      /ObjectType:{E45795B2-9455-11D1-AEBD-0000F80367C1}
9:                      /InheritedObjectType:{BF967ABA-0DE6-11D0-A285-00AA003049E2}
10:                     /AddAce+ /ADSI+
```

As usual, the switch parameters can be taken from a deciphering output or from the various tables related to the Active Directory security descriptors (Tables 4.14 through 4.19).

Before moving to the next security descriptor type, it is interesting to note that the script offers limited support about the GUID numbers management. Actually, it is possible to extend the script in such a way that it accepts the Extended Rights and Active Directory classes display names instead of those ugly GUID numbers. Based on these names, it is always possible to retrieve their corresponding GUID numbers. This will certainly make the script easier to use. However, this logic must be implemented by performing some LDAP search operations on top of ADSI. Since we are focusing on the WMI scripting techniques, this ADSI scripting logic is beyond the scope of this book. However, this could represent a nice extension to have for a day-to-day use of the script.

4.11.4.5.4 The Exchange 2000 mailbox ACE AccessMask property

When an Exchange 2000 mailbox is created, the mailbox security descriptor is initially stored in the **msExchMailboxSecurityDescriptor** attribute of the Active Directory **user** object. The **msExchMailboxSecurityDescriptor** attribute can be accessed via ADSI or WMI, but, again, the *ACE AccessMask* deciphering technique is independent of the access method and the object model representing the security descriptor. Even if the mailbox security descriptor is stored in Active Directory, the deciphering technique is much simpler than the deciphering technique used for an Active Directory object security descriptor. Sample 4.38 uses the same logic as any other standard rights but with a different set of flags.

Sample 4.38 *Deciphering the ACE AccessMask property for Exchange 2000 mailboxes*

```
...:
...:
...:
319:
320:          Case cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
321:               cExchange2000MailboxViaCDOEXM
322:               If (intACEMask And E2K_MB_FULL_MB_ACCESS) Then
323:                  strTemp = strTemp & "," & "E2K_MB_FULL_MB_ACCESS"
324:               End If
325:               If (intACEMask And E2K_MB_SEND_AS) Then
326:                  strTemp = strTemp & "," & "E2K_MB_SEND_AS"
327:               End If
328:               If (intACEMask And E2K_MB_EXTERNAL_ACCOUNT) Then
329:                  strTemp = strTemp & "," & "E2K_MB_EXTERNAL_ACCOUNT"
330:               End If
331:               If (intACEMask And E2K_MB_DELETE) Then
332:                  strTemp = strTemp & "," & "E2K_MB_DELETE"
333:               End If
334:               If (intACEMask And E2K_MB_READ_PERMISSIONS) Then
335:                  strTemp = strTemp & "," & "E2K_MB_READ_PERMISSIONS"
336:               End If
337:               If (intACEMask And E2K_MB_CHANGE_PERMISSIONS) Then
338:                  strTemp = strTemp & "," & "E2K_MB_CHANGE_PERMISSIONS"
339:               End If
340:               If (intACEMask And E2K_MB_TAKE_OWNERSHIP) Then
341:                  strTemp = strTemp & "," & "E2K_MB_TAKE_OWNERSHIP"
342:               End If
343:
...:
...:
...:
```

Table 4.20 *The Exchange 2000 Mailbox ACE AccessMask Values*

Granted & denied rights		Standard View					
		Delete Mailbox storage	Read permissions	Change permissions	Take ownership	Full mailbox access	Associated external account
ACEType							
ADS_ACETYPE_ACCESS_ALLOWED	0x0						
ADS_ACETYPE_ACCESS_DENIED	0x1	X	X	X	X	X	X
ADS_ACETYPE_SYSTEM_AUDIT	0x2						
ACEMask							
E2K_MB_CHANGE_PERMISSIONS	0x40000			X			
E2K_MB_DELETE	0x10000	X					
E2K_MB_EXTERNAL_ACCOUNT	0x4						X
E2K_MB_FULL_MB_ACCESS	0x1					X	X
E2K_MB_READ_PERMISSIONS	0x20000		X				
E2K_MB_SEND_AS	0x2						
E2K_MB_TAKE_OWNERSHIP	0x80000				X		

Table 4.21 *The Exchange 2000 Mailbox ACE Flags Values*

Inheritance & Audit		This object only	Inherit only	This object and subcontainers	This object and children objects	Subcontainers only	Children objects only	This object, subcontainers, and children objects	Subcontainers and children objects	Audit Successful access	Audit Failed access
ACEFlags											
NONE	0x0	X									
ADS_ACEFLAG_OBJECT_INHERIT_ACE[2]	0x1				X		X	X	X		
ADS_ACEFLAG_CONTAINER_INHERIT_ACE[2]	0x2			X		X		X	X		
ADS_ACEFLAG_INHERIT_ONLY_ACE	0x8		X			X	X		X		
ADS_ACEFLAG_INHERITED_ACE[1]	0x10										
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	0x4										
ADS_ACEFLAG_VALID_INHERIT_FLAGS[1]	0x1F		X	X	X	X	X	X	X		
ADS_ACEFLAG_SUCCESSFUL_ACCESS	0x40									X	
ADS_ACEFLAG_FAILED_ACCESS	0x80										X

(1) can only be set by the system.

(2) These two values are not defined in the ADS_ACEFLAG_ENUM. The ADS_ACEFLAG_CONTAINER_INHERIT_ACE is actually defined as the ADS_ACEFLAG_INHERIT_ACE value (0x2). The ADS_ACEFLAG_OBJECT_INHERIT_ACE value is not defined but the 0x1 value is required to correctly decipher the Exchange 2000 ACE inheritance.

The Exchange 2000 mailbox flags with their corresponding user interface settings are summarized in Table 4.20.

Table 4.21 lists the *ACE Flags* to define ACE inheritance for an Exchange 2000 mailbox.

The mailbox security settings shown in Figure 4.30 can be deciphered with the following command line. The output would be as follows:

```
C:\>WMIManageSD.Wsf /E2KMailbox:"CN=LISSOIR Alain,CN=Users,..." /Decipher+ /ADSI+
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Reading Exchange 2000 mailbox security descriptor via ADSI from 'LDAP://CN=LISSOIR...'.

+- ADSI Security Descriptor ----------------------------------------------------------------
| Owner: ............................... LISSWARENET\Alain.Lissoir
| Group: ............................... LISSWARENET\Alain.Lissoir
| Revision: ............................ 1
| Control: ............................. &h8004
                                         SE_DACL_PRESENT
                                         SE_SELF_RELATIVE
|+- ADSI DiscretionaryAcl ---------------------------------------------------------------
||+- ADSI ACE ------------------------------------------------------------------------------
||| AccessMask: ........................ &h20003
                                         E2K_MB_FULL_MB_ACCESS
```

```
18:                                                 E2K_MB_SEND_AS
19:                                                 E2K_MB_READ_PERMISSIONS
20:    ||| AceFlags: ........................... &h2
21:                                                 ADS_ACEFLAG_CONTAINER_INHERIT_ACE
22:                                                 ADS_ACEFLAG_VALID_INHERIT_FLAGS
23:    ||| AceType: ............................ &h0
24:                                                 ADS_ACETYPE_ACCESS_ALLOWED
25:    ||| AceFlagType: ........................ &h0
26:    ||| Trustee: ............................ NT AUTHORITY\SELF
27:    ||+-------------------------------------------------------------------------------------------
28:    |+--------------------------------------------------------------------------------------------
29:    +---------------------------------------------------------------------------------------------
```

Lines 15 through 26 show the ACE configuration for the security settings shown in Figure 4.30. The *ACE AccessMask* is composed of the flags listed in Table 4.20, while the *ACE Flags* property is made up of the flags from Table 4.21.

To customize the Exchange 2000 security descriptor, as shown in Figure 4.30, the following command line can be used:

```
1:   C:\>WMIManageSD.Wsf /E2KMailbox:"CN=LISSOIR Alain,CN=Users,DC=LissWare,DC=Net"
2:                       /Trustee:"NT AUTHORITY\SELF"
3:                       /ACEType:ADS_ACETYPE_ACCESS_ALLOWED
4:                       /ACEMask:E2K_MB_FULL_MB_ACCESS,
5:                                E2K_MB_SEND_AS,
6:                                E2K_MB_READ_PERMISSIONS
7:                       /ACEFlags:ADS_ACEFLAG_CONTAINER_INHERIT_ACE
8:                       /AddAce+ /ADSI+
```

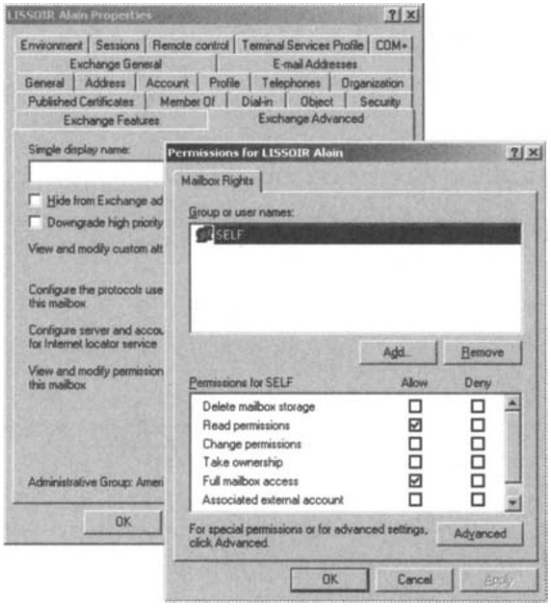


Figure 4.30 *The default Exchange 2000 mailbox security just after creation from the MMC.*

It is important to note that the script sets the security on the mailbox. It doesn't create a mail-enabled or mailbox-enabled Active Directory object. In this example, we use the ADSI security descriptor access method. However, as we have seen in section 4.4.4 ("Which access technique to use? Which security descriptor representation do we obtain?"), the WMI and CDOEXM method can be used as well. The access method depends on certain conditions, which we will discuss in section 4.13.4 ("Updating Exchange 2000 mailbox").

4.11.4.5.5 The registry key ACE AccessMask property

The registry *ACE AccessMask* deciphering technique is no more complicated than any other *ACE AccessMask*. It follows the same coding and deciphering rules as seen previously. As usual, the set of flags to use to decipher the *ACE AccessMask* is dedicated to the registry. Table 4.22 summarizes the various user interface settings possible, with their corresponding values.

Table 4.23 shows the *ACE Flags* used to control the ACE inheritance in a registry hive.

Table 4.22 *The Registry Key ACE AccessMask Values*

Granted & denied rights		Standard View		Advanced View									
		Read	Full Control	Query Value	Set Value	Create Subkey	Enumerate Subkeys	Notify	Create Link	Delete	Write DAC	Write Owner	Read Control
ACEType													
ADS_ACETYPE_ACCESS_ALLOWED	0x0	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]	X[1]
ADS_ACETYPE_ACCESS_DENIED	0x1												
ADS_ACETYPE_SYSTEM_AUDIT	0x2												
ACEMask													
REG_GENERIC_FULL_CONTROL	0xF003F		X										
REG_GENERIC_READ	0x20019	X											
REG_CREATE_LINK	0x20		X						X				
REG_CREATE_SUBKEYS	0x4		X			X							
REG_DELETE	0x10000		X							X			
REG_ENUMERATE_SUBKEYS	0x8	X	X				X						
REG_NOTIFY	0x10	X	X					X					
REG_QUERY_VALUE	0x1	X	X	X									
REG_READ_CONTROL	0x20000	X	X										X
REG_SET_VALUE	0x2		X		X								
REG_WRITE_DAC	0x40000		X								X		
REG_WRITE_OWNER	0x80000		X									X	

(1) **Windows NT 4.0/Windows 2000:** The **ADsSecurity.DLL** from the ADSI Resource Kit does not retrieve the SACL object from the registry.
Windows XP/Windows Server 2003: Unfortunately, a bug in the ADsSecurityUtility interface prevents the retrieval of the SystemACL. Microsoft doesn't plan to fix this bug in the RTM code for timing issues. WMI offers an acceptable work-around for file and folders only. For the registry key, there is no work-around available unless you use the UserRight.Control developed to work around this problem. (See section 4.7.1.2, "Retrieving file and folder security descriptors with ADSI.")

Table 4.23 *The Registry Key ACE Flags Values*

Inheritance		This key only	This key and subkeys	Subkeys only
ACEFlags				
NONE	0x0	X		
CONTAINER_INHERIT_ACE	0x2		X	X
INHERIT_ONLY_ACE	0x8			X

Based on the flags of Table 4.22, Sample 4.39 deciphers the *ACE AccessMask* property. There is nothing new to explain about the logic, since the coding technique remains the same.

Sample 4.39 *Deciphering the ACE AccessMask property for registry keys*

```
...:
...:
...:
343:
344:          Case cRegistryViaADSI
345:               If (intACEMask = REG_GENERIC_FULL_CONTROL) Then
346:                  strTemp = strTemp & "," & "(REG_GENERIC_FULL_CONTROL)"
347:               End If
348:               If (intACEMask = REG_GENERIC_READ) Then
349:                  strTemp = strTemp & "," & "(REG_GENERIC_READ)"
350:               End If
351:
352:               If (intACEMask And REG_QUERY_VALUE) Then
353:                  strTemp = strTemp & "," & "REG_QUERY_VALUE"
354:               End If
355:               If (intACEMask And REG_SET_VALUE) Then
356:                  strTemp = strTemp & "," & "REG_SET_VALUE"
357:               End If
358:               If (intACEMask And REG_CREATE_SUBKEYS) Then
359:                  strTemp = strTemp & "," & "REG_CREATE_SUBKEYS"
360:               End If
...:
379:               If (intACEMask And REG_WRITE_OWNER) Then
380:                  strTemp = strTemp & "," & "REG_WRITE_OWNER"
381:               End If
382:
...:
...:
...:
```

If we take the configuration settings of Figure 4.31, the script output obtained is as follows:

```
  1:  C:\>WMIManageSD.Wsf /RegistryKey:HKLM\SYSTEM\CurrentControlSet\Services\SNMP /Decipher+
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:  Reading registry security descriptor via ADSI from 'HKLM\SYSTEM\CurrentContr...
  6:
  7:  +- ADSI Security Descriptor ----------------------------------------------------------------
  8:  | Owner: ............................... BUILTIN\Administrators
  9:  | Group: ............................... NT AUTHORITY\SYSTEM
 10:  | Revision: ............................ 1
 11:  | Control: ............................. &h8404
 12:                                          SE_DACL_PRESENT
 13:                                          SE_DACL_AUTO_INHERITED
 14:                                          SE_SELF_RELATIVE
 15:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
 16:  ||+- ADSI ACE ---------------------------------------------------------------------------
 17:  ||| AccessMask: ........................ &h30019
 18:                                          REG_QUERY_VALUE
 19:                                          REG_ENUMERATE_SUBKEYS
 20:                                          REG_NOTIFY
 21:                                          REG_DELETE
 22:                                          REG_READ_CONTROL
 23:  ||| AceFlags: ........................... &h2
 24:                                          CONTAINER_INHERIT_ACE
 25:  ||| AceType: ............................ &h0
 26:                                          ACCESS_ALLOWED_ACE_TYPE
 27:  ||| AceFlagType: ........................ &h0
 28:  ||| Trustee: ............................ LISSWARENET\Alain.Lissoir
 29:  ||+-------------------------------------------------------------------------------------------
...:
153:  |+-----------------------------------------------------------------------------------------
154:  +-------------------------------------------------------------------------------------------
```

The highlighted trustee in Figure 4.31 has a full read access to the registry hive. It is also able to delete registry keys below the selected hive. The ACE is deciphered from line 16 through 29 with the flag values of Tables 4.22 and 4.23. To configure the same ACE with the **WMIManageSD.Wsf** script, the following command line must be used:

```
 1:  C:\>WMIManageSD.Wsf /RegistryKey:HKLM\SYSTEM\CurrentControlSet\Services\SNMP
 2:                      /Trustee:LissWareNET\Alain.Lissoir
 3:                      /ACEType:ACCESS_ALLOWED_ACE_TYPE
 4:                      /ACEMask:REG_QUERY_VALUE,
 5:                               REG_ENUMERATE_SUBKEYS,
 6:                               REG_NOTIFY,
 7:                               REG_DELETE,
 8:                               REG_READ_CONTROL
 9:                      /ACEFlags:CONTAINER_INHERIT_ACE
10:                      /AddAce+ /ADSI+
```

The only access method available to read and update the security descriptor is exposed by ADSI. Therefore, the **/ADSI+** switch must be specified in this example.

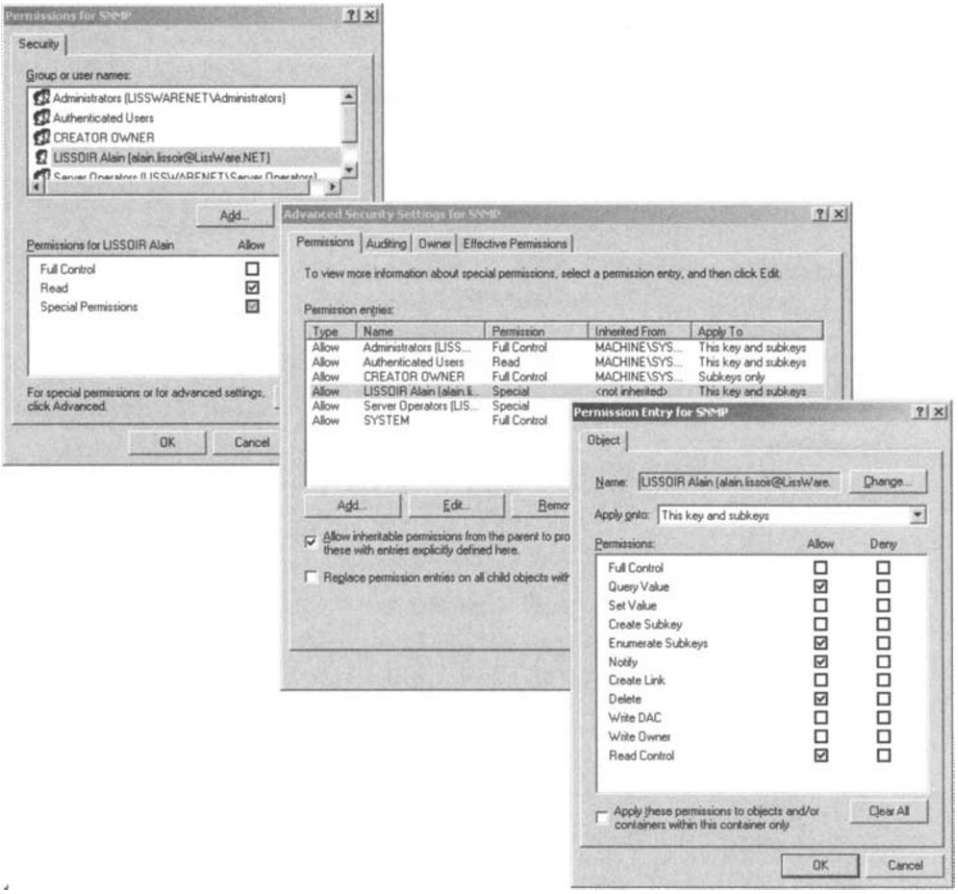


Figure 4.31 *The registry hive security descriptor user interface.*

4.11.4.5.6 The CIM repository namespace ACE AccessMask property

Deciphering the *ACE AccessMask* of a CIM repository namespace is the same as deciphering any other *ACE AccessMask*. Only the flag values are different. Table 4.24 lists the possible configuration settings.

Regarding the *ACE Flags* to configure the ACE inheritance, you can refer to Table 4.25.

As an example, Figure 4.32 shows the default security settings of the **Root\CIMv2** namespace.

Table 4.24 *The CIM Repository Namespace Key ACE AccessMask Values*

Granted & denied rights		Advanced View							
		Execute methods	Full Write	Partial Write	Provider Write	Enable Account	Remote Enable	Read Security	Edit Security
ACEType									
ADS_ACETYPE_ACCESS_ALLOWED	0x0	X	X	X	X	X	X	X	X
ADS_ACETYPE_ACCESS_DENIED	0x1								
ADS_ACETYPE_SYSTEM_AUDIT	0x2	N/A[1]							
ACEMask									
WBEM_ENABLE	0x1					X			
WBEM_FULL_WRITE_REP	0x4		X						
WBEM_METHOD_EXECUTE	0x2	X							
WBEM_PARTIAL_WRITE_REP	0x8		X	X					
WBEM_READ_CONTROL	0x20000							X	
WBEM_REMOTE_ACCESS	0x20						X		
WBEM_WRITE_DAC	0x40000								X
WBEM_WRITE_PROVIDER	0x10		X		X				

(1) SACL is not supported in the WMI CIM repository.

By using the script, the DecipherACEMask() function executes the code segment shown in Sample 4.40.

Sample 4.40 *Deciphering the ACE AccessMask property for CIM repository namespaces*

```
...:
...:
...:
382:
383:          Case cWMINameSpaceViaWMI
384:               If (intACEMask And WBEM_ENABLE) Then
385:                  strTemp = strTemp & "," & "WBEM_ENABLE"
386:               End If
387:               If (intACEMask And WBEM_METHOD_EXECUTE) Then
388:                  strTemp = strTemp & "," & "WBEM_METHOD_EXECUTE"
389:               End If
390:               If (intACEMask And WBEM_FULL_WRITE_REP) Then
391:                  strTemp = strTemp & "," & "WBEM_FULL_WRITE_REP"
392:               End If
393:               If (intACEMask And WBEM_PARTIAL_WRITE_REP) Then
394:                  strTemp = strTemp & "," & "WBEM_PARTIAL_WRITE_REP"
395:               End If
396:               If (intACEMask And WBEM_WRITE_PROVIDER) Then
397:                  strTemp = strTemp & "," & "WBEM_WRITE_PROVIDER"
398:               End If
399:               If (intACEMask And WBEM_REMOTE_ACCESS) Then
400:                  strTemp = strTemp & "," & "WBEM_REMOTE_ACCESS"
401:               End If
402:               If (intACEMask And WBEM_WRITE_DAC) Then
403:                  strTemp = strTemp & "," & "WBEM_WRITE_DAC"
404:               End If
405:               If (intACEMask And WBEM_READ_CONTROL) Then
406:                  strTemp = strTemp & "," & "WBEM_READ_CONTROL"
407:               End If
408:
409:          Case cRegistryViaWMI, cWMINameSpaceViaADSI
410:
```

```
411:          Case Else
412:
413:    End Select
414:
415:    DecipherACEMask = ConvertStringInArray (strTemp, ",")
416:
417:End Function
```

Table 4.25 *The CIM Repository Namespace Key ACE Flags Values*

Inheritance		This namespace only	This namespace and subnamespaces	Subnamespaces only
ACEFlags				
NONE	0x0	X		
CONTAINER_INHERIT_ACE	0x2		X	X
INHERIT_ONLY_ACE	0x8			X

As a result, the right settings in Figure 4.32 are deciphered as follows:

```
 1:  C:\>WMIManageSD.Wsf /WMINameSpace:Root\CIMv2 /Decipher+
 2:  Microsoft (R) Windows Script Host Version 5.6
 3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:  Reading CIM repository namespace security descriptor via WMI from 'Root\CIMv2'.
 6:
 7:  +- ADSI Security Descriptor ----------------------------------------------------------------
 8:  | Owner: ............................... BUILTIN\Administrators
 9:  | Group: ............................... BUILTIN\Administrators
10:  | Revision: ............................ 1
11:  | Control: ............................. &h8004
12:                                          SE_DACL_PRESENT
13:                                          SE_SELF_RELATIVE
14:  |+- ADSI DiscretionaryAcl ------------------------------------------------------------------
15:  ||+- ADSI ACE ------------------------------------------------------------------------------
16:  ||| AccessMask: ........................ &h1F
17:                                          WBEM_ENABLE
18:                                          WBEM_METHOD_EXECUTE
19:                                          WBEM_FULL_WRITE_REP
20:                                          WBEM_PARTIAL_WRITE_REP
21:                                          WBEM_WRITE_PROVIDER
22:  ||| AceFlags: ......................... &hA
23:                                          CONTAINER_INHERIT_ACE
24:                                          INHERIT_ONLY_ACE
25:  ||| AceType: ......................... &h0
26:                                          ACCESS_ALLOWED_ACE_TYPE
27:  ||| AceFlagType: ...................... &h0
28:  ||| Trustee: ......................... LISSWARENET\Alain.Lissoir
29:  ||+-------------------------------------------------------------------------------------------
..:
61:  |+-------------------------------------------------------------------------------------------
62:  +---------------------------------------------------------------------------------------------
```

Figure 4.32 *The Root\CIMv2 namespace security descriptor user interface.*

From line 15 through 29, the ACE for the highlighted trustee in Figure 4.32 is deciphered with the flags coming from Tables 4.24 and 4.25. To configure the same ACE with the **WMIManageSD.Wsf** script, the following command line must be used:

```
1:   C:\>WMIManageSD.Wsf /WMINameSpace:Root\CIMv2
2:                       /Trustee:LissWareNET\Alain.Lissoir
3:                       /ACEType:ACCESS_ALLOWED_ACE_TYPE
4:                       /ACEMask:WBEM_ENABLE,
5:                                WBEM_METHOD_EXECUTE,
6:                                WBEM_FULL_WRITE_REP,
7:                                WBEM_PARTIAL_WRITE_REP,
8:                                WBEM_WRITE_PROVIDER
9:                       /ACEFlags:CONTAINER_INHERIT_ACE,INHERIT_ONLY_ACE
10:                      /AddAce+
```

The only access method available to read and update the security descriptor is exposed by WMI via the *GetSD* and *SetSD* methods of the *__SystemSecurity* singleton system class (see sections 4.7.6 and 4.13.6). Therefore, no /ADSI+ switch is specified in this example.

4.12 Modifying the security descriptor

In the previous sections we have seen how to access a security descriptor on a secured entity and how to decipher the security descriptor. Although we saw some of the command-line parameters to use to update the security descriptor based on deciphering results and various tables, we did not discover the scripting technique to effectively modify the various security descriptor components. The purpose of this section is to explain how the security descriptor modification is coded, based on the parameters given on the command line.

4.12.1 Updating the security descriptor Owner

The scripting technique used to update the *Owner* property of the security descriptor differs, based on the security descriptor object model used. With the ADSI object model, the owner is nothing other than a single property containing the "Domain\UserID" owner. However, with the WMI object model, the owner is an instance of the *Win32_Trustee* class and requires some extra logic to create the instance. Sample 4.41 shows the SetSDOwner() function, which includes the script code for the two object models.

Sample 4.41 *Updating the security descriptor owner*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function SetSDOwner(strSIDResolutionDC, strUserID, strPassword, _
10:                    objSD, strTrustee, intSDType)
..:
14:    Select Case intSDType
15:' Here we have an ADSI security descriptor representation
16:          Case cFileViaADSI, _
17:               cShareViaADSI, _
18:               cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
19:               cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
20:               cExchange2000MailboxViaCDOEXM, _
21:               cRegistryViaADSI, _
22:               cWMINameSpaceViaWMI
23:
24:               objSD.Owner = strTrustee
25:
```

```
26:' Here we have a WMI security descriptor representation
27:          Case cFileViaWMI, _
28:               cShareViaWMI
29:
30:               objSD.Owner = CreateTrustee (strTrustee, _
31:                                            strSIDResolutionDC, _
32:                                            strUserID, _
33:                                            strPassword)
34:
35:' Here we can't retrieve a security descriptor via this access method.
36:          Case cRegistryViaWMI, cWMINameSpaceViaADSI
37:
38:    End Select
39:
40:    WScript.Echo "Security descriptor owner updated to '" & strTrustee & "'."
41:
42:    Set SetSDOwner = objSD
43:
44:End Function
```

From line 16 through 24, the owner is stored in the *Owner* property of the ADSI **SecurityDescriptor** object (line 24). With the ADSI object model, the update of the security descriptor owner is pretty straightforward, since it assigns a literal string to the *Owner* property.

To update the security descriptor owner with the WMI object model (lines 27 through 33), the script must first create an instance of the *Win32_Trustee* class. This instance is created in the CreateTrustee() function (lines 30 through 33). The CreateTrustee() function is shown in Sample 4.42.

Sample 4.42 *Creating a Win32_Trustee instance*

```
 .:
 .:
 .:
 8:' -------------------------------------------------------------------------------------------
 9:Function CreateTrustee (strTrustee, strSIDResolutionDC, strUserID, strPassword)
..:
17:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
18:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
19:
20:    Set objWMIServices = objWMILocator.ConnectServer(strSIDResolutionDC, ""Root\CIMv2"", _
21:                                                     strUserID, strPassword)
..:
24:    Set objTrustee = objWMIServices.Get("Win32_Trustee").SpawnInstance_()
25:
26:    objTrustee.Domain = ExtractUserDomain (strTrustee)
27:    objTrustee.Name = ExtractUserID (strTrustee)
28:
29:    Set objWMIInstances = objWMIServices.ExecQuery ("Select SID From Win32_Account Where " & _
30:                                                    "Name='" & objTrustee.Name & "'")
..:
33:    If objWMIInstances.Count = 1 Then
34:       For Each objWMIInstance In objWMIInstances
```

```
35:          objTrustee.SIDString = objWMIInstance.SID
36:
37:          Set objSID = objWMIServices.Get("Win32_SID.SID='" & objWMIInstance.SID & "'")
38:
39:          objTrustee.SID = objSID.BinaryRepresentation
40:          objTrustee.SidLength = objSID.SidLength
..:
43:       Next
44:    Else
45:       WScript.Echo "WMI Trustee '" & strTrustee & "' not found on SIDResolutionDC '" & _
46:                    strSIDResolutionDC & "'."
47:       WScript.Quit (1)
48:    End If
..:
54:    Set CreateTrustee = objTrustee
..:
58:End Function
```

This function is slightly more complex, because an instance of the *Win32_Trustee* class requires the SID of the user. Because the SID of the user must be resolved against the domain hosting that user, the CreateTrustee() function requires some extra parameters:

- **Trustee:** This parameter contains the "Domain\UserID" for which the *Win32_Trustee* must be created.
- **SIDResolutionDC:** This parameter contains the Fully Qualified Domain Name (FQDN) of the domain controller to be used for the SID resolution. It should be a domain controller hosting the domain name given in the trustee. By default, if no SIDResolutionDC is given, the localhost is used. If the script is executed on a system not hosting the trustee in its local SAM (or domain if it is a domain controller), the **/SIDResolutionDC** parameter must be given. In such a case, it must be the FQDN of the domain controller able to resolve the SID for the specified "Domain\UserID." If the trustee could not be resolved to a SID, the *Win32_Trustee* instance creation will fail. So, make sure that the selected CIM repository (the default one or the one specified by the **/SIDResolutionDC** switch) always knows the passed "UserID" for the associated "Domain." Note that when the security descriptor is represented in the ADSI object model, the SID resolution is performed inside the ADSI COM object managing the trustee, which makes the task a little bit easier, but this requires that the workstation is well connected to the right domain to resolve the trustee (directly or via trusts).
- **UserID:** This parameter is the UserID that WMI uses for the WMI remote connection. It corresponds to the UserID given with the **/User** switch. If the switch is not specified, the security context of the user

executing the script is used to perform the WMI connection to the SIDResolutionDC. In such a case, you must make sure that this user has access, via WMI, to the required system in the **Root\CIMv2** namespace.

- **Password:** This parameter is the password to be used with the UserID performing the WMI remote connection.

From line 17 through 21, the script performs a WMI connection to resolve the trustee to a SID. Once completed, a new instance of the *Win32_Trustee* class is created (line 24). Next, the domain name (line 26) and the user name (line 27) of the trustee are assigned to the corresponding *Win32_Trustee* class properties. To find the SID corresponding to the user name, the script performs a WQL query (lines 29 and 30). If there is one valid response from the WQL query (line 33), the result is enumerated (lines 34 through 43), and the string representation of the SID is saved in the *Win32_Trustee* instance (line 35). In case of a SID resolution problem, the script displays an error message and stops its execution (lines 44 through 48).

Because a *Win32_Trustee* instance also requires a binary representation of the SID, lines 37 through 40 retrieve some extra SID information with another WQL query (line 37). This query is based on the *Win32_SID* class, which uses the SID string representation as one of its properties. Once found, the script assigns the remaining *Win32_Trustee* properties with the binary SID information (lines 39 and 40). Next, the CreateTrustee() function returns the *Win32_Trustee* instance, which is assigned to the *Owner* property of the *Win32_SecurityDescriptor* instance at line 30 of Sample 4.41.

To produce the result shown in Figure 4.33, the command line to use to update the security descriptor owner is as follows:

```
1:  C:\>WMIManageSD.Wsf /ADObject:"CN=LISSOIR Alain,CN=Users,DC=LissWare,DC=Net"
2:                      /Owner:LissWareNET\Alain.Lissoir /ADSI+
```

4.12.2 Updating the security descriptor Group

The logic used to update the *Group* property is the same as that used to update the *Owner* property (see Sample 4.43). Via ADSI, the *Group* property is directly assigned with the "Domain\Group" value (line 24). Via WMI, Sample 4.43 makes use of the CreateTrustee() function previously explained (see Sample 4.42).

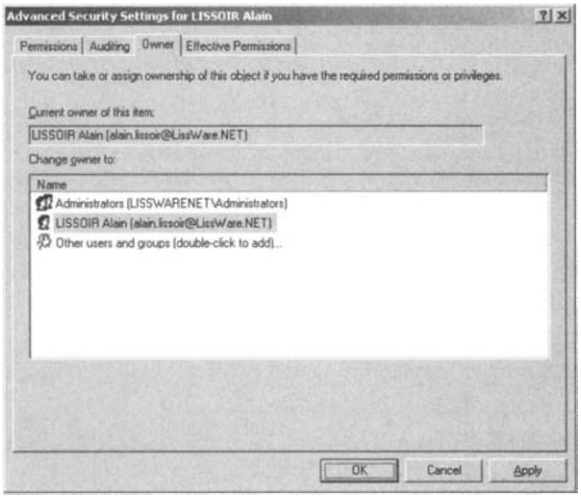


Figure 4.33 *An Active Directory security descriptor owner.*

Sample 4.43 *Updating the security descriptor group*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------------
 9:Function SetSDGroup(strSIDResolutionDC, strUserID, strPassword, _
10:                    objSD, strTrustee, intSDType)
..:
14:    Select Case intSDType
15:' Here we have an ADSI security descriptor representation
16:           Case cFileViaADSI, _
17:                cShareViaADSI, _
18:                cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
19:                cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
20:                cExchange2000MailboxViaCDOEXM, _
21:                cRegistryViaADSI, _
22:                cWMINameSpaceViaWMI
23:
24:                objSD.Group = strTrustee
25:
26:' Here we have a WMI security descriptor representation
27:           Case cFileViaWMI, _
28:                cShareViaWMI
29:
30:                objSD.Group = CreateTrustee (strTrustee, _
31:                                             strSIDResolutionDC, _
32:                                             strUserID, _
33:                                             strPassword)
34:
35:' Here we can't retrieve a security descriptor via this access method.
36:           Case cRegistryViaWMI, _
37:                cWMINameSpaceViaADSI
38:
39:    End Select
```

```
40:
41:    WScript.Echo "Security descriptor group updated to '" & strTrustee & "'."
42:
43:    Set SetSDGroup = objSD
44:
45:End Function
```

4.12.3 Updating the security descriptor Control Flags

The security descriptor *Control Flags* property contains several flags (see Table 4.8, "The security descriptor *Control Flags* values"). However, the script does not allow the modification of all flags, since the script already manages some of them internally. The flags that can be specified from the command line are:

- SE_DACL_PROTECTED
- SE_SACL_PROTECTED
- SE_DACL_AUTO_INHERIT_REQ
- SE_SACL_AUTO_INHERIT_REQ

As shown in Table 4.8, these flags are controlling the security descriptor behavior regarding the inherited ACE. For instance, if the security descriptor is protected against inherited ACE, which means that inherited ACE is not inherited by the security descriptor (SE_DACL_PROTECTED=ON), we have a configuration similar to the one shown in Figure 4.34 (right pane). Note that the user interface check box is unchecked when the SE_DACL_PROTECTED flag is turned ON.

During a *Control Flags* modification (SE_DACL_PROTECTED=OFF to SE_DACL_PROTECTED=ON, or as shown in Figure 4.34, left and

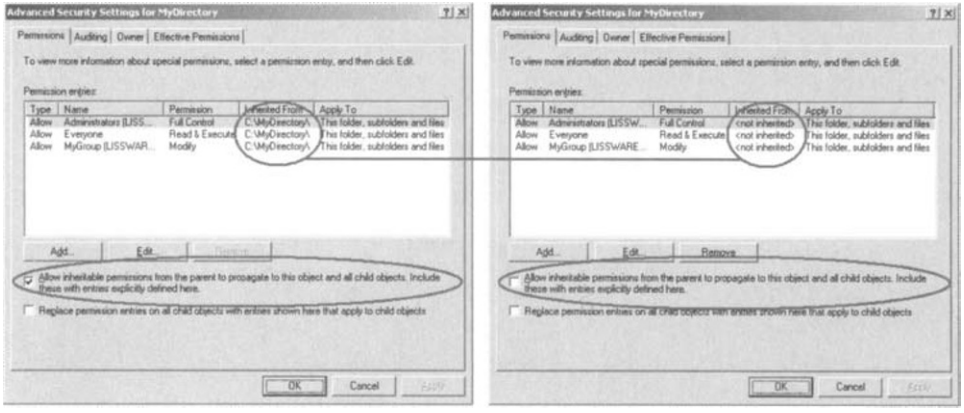


Figure 4.34 *The Control Flags configuration.*

right panes) by script via ADSI or WMI, the inherited ACEs are not inherited anymore and become directly applied ACE (or noninherited ACE).

The next command line turns to ON the SE_DACL_PROTECTED flag, since it is currently turned OFF (left pane of Figure 4.34):

```
1:  C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /SDControls:SE_DACL_PROTECTED
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.
6:  ACL inheritance protection is OFF and will be turned ON.
7:  Setting File or Folder security descriptor via WMI to 'C:\MyDirectory'.
```

Now the SE_DACL_PROTECTED is turned ON (right pane of Figure 4.34).

If the script is run a second time with the same parameters, the bit corresponding to the SE_DACL_PROTECTED flag is switched back to OFF.

```
1:  C:\>WMIManageSD.Wsf /FileSystem:C:\MyDirectory /SDControls:SE_DACL_PROTECTED
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Reading File or Folder security descriptor via WMI from 'C:\MyDirectory'.
6:  ACL inheritance protection is ON and will be turned OFF.
7:  Setting File or Folder security descriptor via WMI to 'C:\MyDirectory'.
```

As a consequence (see Figure 4.35), the inherited ACEs are added to the noninherited ACEs.

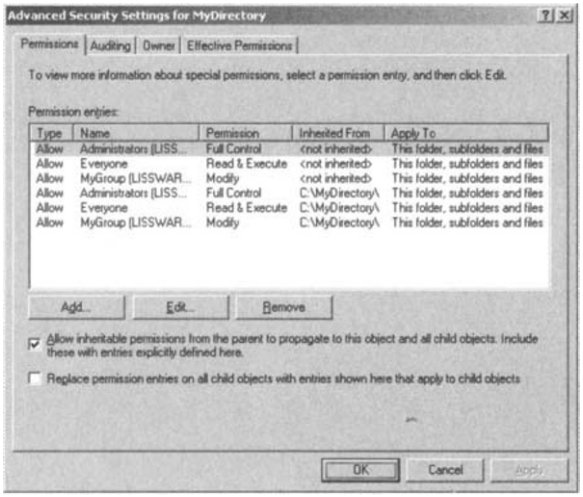


Figure 4.35 *The effect of resetting the SE_DACL_PROTECTED flag twice.*

The manipulation of all *Control Flags* bits is basically the same. Whenever the security descriptor is represented in the ADSI or WMI object models, Sample 4.44 uses the same logic as previously. Because the *Control Flags* property is named *Control* in the ADSI object model and named *ControlFlags* in the WMI object model, the script code must take the object model used into consideration to assign the new value, despite the fact the logic is the same (lines 23 through 27 for the ADSI object model, lines 35 through 39 for the WMI object model).

Sample 4.44 *Updating the security descriptor control flags*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function SetSDControlFlags(objSD, intSDControlFlags, intSDType)
..:
13:    Select Case intSDType
14:' Here we have an ADSI security descriptor representation
15:          Case cFileViaADSI, _
16:               cShareViaADSI, _
17:               cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
18:               cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
19:               cExchange2000MailboxViaCDOEXM, _
20:               cRegistryViaADSI, _
21:               cWMINameSpaceViaWMI
22:
23:               If (objSD.Control And intSDControlFlags) = intSDControlFlags Then
24:                  WScript.Echo "ACL inheritance protection is OFF and will be turned ON."
25:               Else
26:                  WScript.Echo "ACL inheritance protection is ON and will be turned OFF."
27:               End If
28:
29:               objSD.Control = objSD.Control Xor intSDControlFlags
30:
31:' Here we have a WMI security descriptor representation
32:          Case cFileViaWMI, _
33:               cShareViaWMI
34:
35:               If (objSD.ControlFlags And intSDControlFlags) = intSDControlFlags Then
36:                  WScript.Echo "ACL inheritance protection is OFF and will be turned ON."
37:               Else
38:                  WScript.Echo "ACL inheritance protection is ON and will be turned OFF."
39:               End If
40:
41:               objSD.ControlFlags = objSD.ControlFlags Xor intSDControlFlags
42:
43:' Here we can't retrieve a security descriptor via this access method.
44:          Case cRegistryViaWMI, _
45:               cWMINameSpaceViaADSI
46:
47:    End Select
48:
49:    Set SetSDControlFlags = objSD
50:
51:End Function
```

4.12.4 Adding an ACE

To manipulate ACEs in security descriptors, it is necessary to look at the ACL. Since ACLs are represented differently in the ADSI and WMI object models, the script must handle them differently. The next two sections will explain the logic used for the ACE additions in the ADSI and WMI object models, respectively. The code snippets are direct applications of sections 4.11.3 ("Deciphering the Access Control Lists") and 4.11.4 ("Deciphering the Access Control Entries") as examples of configuring ACEs to make use of the **/AddACE+** switch.

The AddACE() function is executed when the **/AddACE+** switch is specified on the command line. For example:

```
C:\>WMIManageSD.Wsf /WMINameSpace:Root\MyNameSpace
                    /Trustee:BUILTIN\Administrators /ACEType:ACCESS_ALLOWED_ACE_TYPE
                    /ACEMask:WBEM_ENABLE,
                             WBEM_METHOD_EXECUTE,
                             WBEM_WRITE_PROVIDER,
                             WBEM_REMOTE_ACCESS
                    /ACEFlags:CONTAINER_INHERIT_ACE
                    /AddAce+
```

4.12.4.1 Adding an ACE in the ADSI object model

The AddACE() function shown in Samples 4.45 and 4.46 implements the logic to add ACEs in the ADSI and WMI object models. The function is divided into two parts: one for the ADSI object model (Sample 4.45) and the second one (Sample 4.46) for the WMI object model. Because the new ACE is built and added in the function, the function exposes several parameters:

- **objWMIServices:** This parameter is an **SWBemServices** object representing the WMI connection to the local CIM repository. This parameter is only used to create a new instance of the *Win32_ACE* class when the security descriptor is represented in the WMI object model. For the ADSI object model, this parameter can be set to Null.
- **SIDResolutionDC, UserID, and Password:** These parameters are the ones used by the CreateTrustee() function (Sample 4.42, "Creating a *Win32_Trustee* instance"). They are only used when the security descriptor is represented in the WMI object model. For the ADSI object model, these parameters can be set to Null (see Sample 4.4, "Connecting to files and folders with ADSI [Part II]," line 591).
- **objSD:** This parameter is the structural representation of the security descriptor in the ADSI or WMI object model.

- **Trustee, ACEType, AccessMask, ACEFlags, ACLType, ObjectType,** and **InheritedObjectType:** These parameters are the new ACE properties. These parameters are used both in the ADSI and WMI object models. Note that the *ObjectType* and *InheritedObjectType* parameters are only used when the security descriptor is an Active Directory security descriptor.
- **SDType:** This parameter determines the security descriptor access method, which implicitly determines the object model.

Sample 4.45 *Adding ACE in the ADSI object model (Part I)*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function AddACE(objWMIServices, strSIDResolutionDC, strUserID, strPassword, _
10:            objSD, strTrustee, intACEType, intAccessMask, intACEFlags, intACLType, _
11:            strObjectType, strInheritedObjectType, intSDType)
..:
21:    Select Case intSDType
22:' Here we have an ADSI security descriptor representation
23:          Case cFileViaADSI, _
24:               cShareViaADSI, _
25:               cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
26:               cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
27:               cExchange2000MailboxViaCDOEXM, _
28:               cRegistryViaADSI, _
29:               cWMINameSpaceViaWMI
30:
31:               Select Case intACLType
32:                      Case cDACL
33:                           Set objACL = objSD.DiscretionaryAcl
34:                      Case cSACL
35:                           Set objACL = objSD.SystemAcl
36:                      Case Else
37:                           Exit Function
38:               End Select
39:
40:               Set objNewACE = CreateObject("AccessControlEntry")
41:
42:               objNewACE.Trustee = strTrustee
43:               objNewACE.AceType = intACEType
44:               objNewACE.AccessMask = intAccessMask
45:               objNewACE.AceFlags = intACEFlags
46:
47:               Select Case intSDType
48:                      Case cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
49:
50:                           If Len (strObjectType) Then
51:                              objNewACE.ObjectType = strObjectType
52:                              objNewACE.Flags = objNewACE.Flags Or _
53:                                                ADS_FLAG_OBJECT_TYPE_PRESENT
54:                           End If
55:
56:                           If Len (strInheritedObjectType) Then
```

```
57:                         objNewACE.InheritedObjectType = strInheritedObjectType
58:                         objNewACE.Flags = objNewACE.Flags Or _
59:                                           ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
60:                     End If
61:
62:               Case cFileViaADSI, _
63:                    cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
64:                    cExchange2000MailboxViaCDOEXM, _
65:                    cRegistryViaADSI, _
66:                    cWMINameSpaceViaWMI
67:
68:          End Select
69:
70:          objACL.AddAce objNewACE
71:
72:          Select Case intACLType
73:                 Case cDACL
74:                      objSD.DiscretionaryAcl = objACL
75:                      objSD.Control = objSD.Control Or SE_DACL_PRESENT
76:                 Case cSACL
77:                      objSD.SystemAcl = objACL
78:                      objSD.Control = objSD.Control Or SE_SACL_PRESENT
79:          End Select
80:
81:          Wscript.Echo "Trustee '" & strTrustee & _
82:                       "' has been added to security descriptor."
..:
..:
..:
```

Sample 4.45 starts by extracting the ACL from the ADSI security descriptor. Based on the **ACLType** parameter, the Discretionary ACL or the System ACL is extracted (lines 31 through 38). Next, it builds the new ACE by creating a new instance of the **AccessControlEntry** ADSI object (line 40) and assigns the ACE properties (lines 42 through 45). If the security descriptor is an Active Directory security descriptor, then the **ObjectType** and/or the **InheritedObjectType** parameters are assigned (lines 47 through 68). These two parameters contain GUID numbers, as explained in sections 4.11.4.5.3.1 ("Understanding the ACE ObjectType property") and 4.11.4.5.3.2 ("Understanding the ACE InheritedObjectType property"). Because this portion of the script manages the ADSI security descriptor representation, it must set the *ACE FlagType* property of the ADSI **AccessControlEntry** object to save the GUID values back to the security descriptor.

Next, the new ACE is added to the ACL (line 70), and the ACL is stored back in the security descriptor according to the **ACLType** parameter (lines 72 through 79).

4.12.4.2 Adding an ACE in the WMI object model

In the WMI object model, the overall logic is exactly the same as the ADSI object model. However, instead of manipulating the ACE from a collection

(ACL), Sample 4.46 manipulates an array containing ACEs. As before, it extracts the ACL from the WMI security descriptor according to the **ACLType** parameter (lines 92 through 99). Next, it creates a new ACE with the use of the **SWBemServices** object (line 101) and assigns the various ACE properties (lines 103 through 109). Note the use of the CreateTrustee() function (Sample 4.42, "Creating a *Win32_Trustee* instance") to assign the WMI ACE *Trustee* property. Once completed, the new ACE is added to the ACL (lines 111 through 119). Next, the ACL is stored back to the security descriptor according to the **ACLType** parameter (lines 123 through 130).

Sample 4.46 *Adding ACE in the WMI object model (Part II)*

```
..:
..:
..:
 87:
 88:' Here we have a WMI security descriptor representation
 89:          Case cFileViaWMI, _
 90:               cShareViaWMI
 91:
 92:               Select Case intACLType
 93:                      Case cDACL
 94:                           arrayACL = objSD.DACL
 95:                      Case cSACL
 96:                           arrayACL = objSD.SACL
 97:                      Case Else
 98:                           Exit Function
 99:               End Select
100:
101:               Set objNewACE = objWMIServices.Get("Win32_ACE").SpawnInstance_()
102:
103:               objNewACE.Trustee = CreateTrustee (strTrustee, _
104:                                                  strSIDResolutionDC, _
105:                                                  strUserID, _
106:                                                  strPassword)
107:               objNewACE.AceType = intACEType
108:               objNewACE.AccessMask = intAccessMask
109:               objNewACE.AceFlags = intACEFlags
110:
111:               ' If ACL already contains some ACE, make sure we don't destroy them ...
112:               If IsArray (arrayACL) Then
113:                  intIndice = UBound(arrayACL)
114:                  ReDim Preserve arrayACL (intIndice + 1)
115:                  Set arrayACL (intIndice + 1) = objNewACE
116:               Else
117:                  ReDim arrayACL (0)
118:                  Set arrayACL (0) = objNewACE
119:               End If
...:
123:               Select Case intACLType
124:                      Case cDACL
125:                           objSD.DACL = arrayACL
126:                           objSD.ControlFlags = objSD.ControlFlags Or SE_DACL_PRESENT
```

```
127:                        Case cSACL
128:                             objSD.SACL = arrayACL
129:                             objSD.ControlFlags = objSD.ControlFlags Or SE_SACL_PRESENT
130:                End Select
131:
132:                Wscript.Echo "Trustee '" & strTrustee & _
133:                             "' has been added to security descriptor."
134:
135:' Here we do not have a security descriptor available via these access methods.
136:           Case cRegistryViaWMI, cWMINameSpaceViaADSI
137:
138:    End Select
...:
142:End Function
```

4.12.5 Removing an ACE

Since the script must manage ACLs in two different object models to remove an ACE from an ACL, the coding technique will be different. Moreover, the logic to remove an ACE from an ACL is totally different from ACE additions. The next two sections cover this functionality in the DelACE() function with the ADSI object model (Sample 4.47) and the WMI object model (Sample 4.48).

The DelACE() function is executed when the **/DelACE+** switch is specified on the command line. For example:

```
C:\>WMIManageSD.Wsf /Share:MyDirectory /Trustee:Everyone /DelAce+
```

4.12.5.1 Removing ACE in the ADSI object model

Basically, the DelACE() input parameters are almost the same as the AddACE() parameters. However, the DelACE() function does not require all ACE details. Only the **Trustee** is necessary. Simply said, the function browses all ACEs in the ACL and removes any ACE that has a matching **Trustee**. Of course, if a trustee is granted with many different rights (which means that it could exist in many different ACEs), all ACEs matching that trustee will be removed. If a more granular removal technique is required, then the match must be made on more ACE properties than on the **Trustee**, which will require more command-line parameters not supported by the **WMIManageSD.Wsf** script in its current version.

Note that it is possible to use a special keyword (**REMOVE_ALL_ACE**) assigned to the **Trustee** parameter of the DelACE() function to remove all ACEs present in the ACL. In such a case, after this removal, the script grants the "Everyone" Full Control right to make sure that the object is still accessible. Granting "Everyone" Full Control is equal to suppressing the concept of controlling the access. From a human point of view, this makes

sense, but from a computer point of view, removing all access controls means no access at all. That's why this trustee is added. Note that this right is only set for a DACL. In the case of a SACL, no default ACE is created. The ACL of a SACL will be emptied.

Now that we understand the DelACE() function features, let's see how the coding works (see Sample 4.47).

Sample 4.47 *Removing ACE in the ADSI object model (Part I)*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function DelACE(objWMIServices, strSIDResolutionDC, strUserID, strPassword, _
10:                objSD, strTrustee, intACLType, intSDType)
..:
..:
..:
26:    Select Case intSDType
27:' Here we have an ADSI security descriptor representation
28:           Case cFileViaADSI, _
29:                cShareViaADSI, _
30:                cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
31:                cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
32:                cExchange2000MailboxViaCDOEXM, _
33:                cRegistryViaADSI, _
34:                cWMINameSpaceViaWMI
35:
36:                Select Case intACLType
37:                       Case cDACL
38:                            Set objACL = objSD.DiscretionaryAcl
39:                       Case cSACL
40:                            Set objACL = objSD.SystemAcl
41:                       Case Else
42:                            Exit Function
43:                End Select
44:
45:                ' Only proceed an ACE removal, if there is at least one available ...
46:                If objACL.AceCount Then
47:                   If Ucase(strTrustee) = "REMOVE_ALL_ACE" Then
48:                      For Each objACE In objACL
49:                          objACL.RemoveAce (objACE)
50:                      Next
51:
52:                      If objACL.AceCount = 0 Then
53:                         Select Case intACLType
54:                                Case cDACL
55:                                     Set objNewACE = CreateObject("AccessControlEntry")
56:
57:                                     objNewACE.Trustee = "Everyone"
58:                                     objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
59:                                     objNewACE.AccessMask = ADS_RIGHT_GENERIC_ALL
60:                                     objNewACE.AceFlags = CONTAINER_INHERIT_ACE Or _
61:                                                          OBJECT_INHERIT_ACE
62:
63:                                     Wscript.Echo "All ACE removed, Trustee '" & _
```

```
64:                                          objNewACE.Trustee & _
65:                                          "' (Full Control) has been created."
66:
67:                              objACL.AddAce objNewACE
68:
69:                              objSD.Control = objSD.Control Or _
70:                                          SE_DACL_PRESENT
...:
73:                          Case cSACL
74:                              objSD.Control = objSD.Control And _
75:                                          (NOT SE_SACL_PRESENT)
76:                      End Select
77:                  End If
78:
79:                  boolRemoveAce = True
80:              Else
81:                  For Each objACE In objACL
82:                      If Ucase(objACE.Trustee) = Ucase(strTrustee) Then
83:                          If (objACE.AceFlags And INHERITED_ACE) = INHERITED_ACE Then
84:                              Wscript.Echo "Trustee '" & objACE.Trustee & _
85:                                          "' is inherited and therefore can't be removed."
86:                          Else
87:                              Wscript.Echo "Trustee '" & strTrustee & _
88:                                      "' has been removed from security descriptor."
89:                              objACL.RemoveAce (objACE)
90:
91:                              If objACL.AceCount = 0 Then
92:                                  Select Case intACLType
93:                                      Case cDACL
94:                                        Set objNewACE = CreateObject("AccessControlEntry")
95:
96:                                        objNewACE.Trustee = "Everyone"
97:                                        objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
98:                                        objNewACE.AccessMask = ADS_RIGHT_GENERIC_ALL
99:                                        objNewACE.AceFlags = CONTAINER_INHERIT_ACE Or _
100:                                                       OBJECT_INHERIT_ACE
101:
102:                                       Wscript.Echo "All ACE removed, Trustee '" & _
103:                                                   objNewACE.Trustee & _
104:                                                   "' (Full Control) has been created."
105:
106:                                       objACL.AddAce objNewACE
107:
108:                                       objSD.Control = objSD.Control Or _
109:                                                       SE_DACL_PRESENT
...:
112:                                    Case cSACL
113:                                      objSD.Control = objSD.Control And _
114:                                                      (NOT SE_SACL_PRESENT)
115:                                 End Select
116:                             End If
117:
118:                             boolRemoveAce = True
119:                         End If
120:                     End If
121:                 Next
122:             End If
123:
124:             If objACL.AceCount = 0 Then
125:                 Wscript.Echo "All ACE removed."
```

```
126:                End If
127:
128:                Select Case intACLType
129:                       Case cDACL
130:                            objSD.DiscretionaryAcl = objACL
131:                       Case cSACL
132:                            objSD.SystemAcl = objACL
133:                End Select
134:             Else
135:                WScript.Echo "No existing ACE to remove."
136:                WScript.Quit(1)
137:             End If
...:
...:
...:
```

As with the AddACE() function, the DelACE() function first retrieves the DACL or the SACL, based on the **ACLType** parameter (lines 36 through 43). Once complete, the DelACE() function is divided into two parts for each object model:

1. **Removing all ACEs (lines 47 through 79):** This part makes use of the **REMOVE_ALL_ACE** keyword to remove all available ACEs. The script enumerates all ACEs to remove instead of creating a new ACL (lines 48 through 50). Recreating a new ACL will force the version number of the ACL to be set in the ADSI object model. To keep the logic totally generic, the script keeps the ACL object intact by removing only ACEs. Since the **AccessControlList** object exposes the *RemoveACE* method, this makes the coding technique quite easy (line 49). Of course, creating a brand new ACL is also a valid technique. Next, if the ACL is emptied (line 52), the script checks if the ACL comes from a DACL (line 54) or a SACL (line 73). In case of a DACL, a default ACE is created (lines 55 through 70). At lines 69 and 70, the script sets the security descriptor *Control Flags* according to the DACL presence. At lines 74 and 75, the script sets the SACL presence to OFF in the security descriptor *Control Flags*, since a default ACE is only created for a DACL. Note that inherited ACEs are not removed by ADSI. They continue to appear as inherited ACEs. If they must be removed, the SE_DACL_PROTECTED flag must be first set to ON, or the parent object defining the inherited ACE must be modified accordingly.

2. **Removing a specific ACE based on the Trustee name (lines 81 through 121):** In such a case, the script enumerates all ACEs (lines 81 through 121), but if the **Trustee** DelACE() function parameter matches the *Trustee* property of an examined ACE (line

82), then the ACE is removed (line 89). Note that inherited ACEs are not removed (lines 83 through 85). Once the ACE is removed, and if the DACL does not contain any ACE, a default ACE based on the "Everyone" trustee is created (lines 93 through 109), as done in the first part of the code. Next, the security descriptor *Control Flags* is set accordingly (lines 108 and 109, lines 113 and 114).

4.12.5.2 Removing ACE in the WMI object model

The ACE removal in the WMI object model follows the same logic as the ACE removal in the ADSI object model. Of course, as mentioned previously, instead of manipulating a collection of ACEs in an ACL, Sample 4.48 manipulates items in an array, where the array represents the ACL and array items represent ACEs.

Sample 4.48 *Removing ACE in the WMI object model (Part II)*

```
...:
...:
...:
140:
141:' Here we have a WMI security descriptor representation
142:          Case cFileViaWMI, _
143:               cShareViaWMI
144:
145:               Select Case intACLType
146:                      Case cDACL
147:                           arrayACL = objSD.DACL
148:                      Case cSACL
149:                           arrayACL = objSD.SACL
150:                      Case Else
151:                           Exit Function
152:               End Select
153:
154:               ' Only proceed an ACE removal, if there is at least one available ...
155:               If IsArray(arrayACL) Then
156:                  If UCase (strTrustee) = "REMOVE_ALL_ACE" Then
157:                     If UBound(arrayACL) = 0 Then
...:
...:
...:
158:             Select Case intACLType
159:                    Case cDACL
160:                         Set objNewACE = objWMIServices.Get("Win32_ACE").SpawnInstance_()
161:
162:                         objNewACE.Trustee = CreateTrustee ("Everyone", _
163:                                                            strSIDResolutionDC, _
164:                                                            strUserID, _
165:                                                            strPassword)
166:                         objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
167:                         objNewACE.AccessMask = ADS_RIGHT_GENERIC_ALL
168:                         objNewACE.AceFlags = CONTAINER_INHERIT_ACE Or _
```

```
                                        OBJECT_INHERIT_ACE

                    Wscript.Echo "All ACE removed, Trustee 'Everyone' " & _
                                 "(Full Control) has been created."

                    ReDim arrayNewACL (0)
                    Set arrayNewACL (0) = objNewACE

                    objSD.ControlFlags = objSD.ControlFlags Or _
                                         SE_DACL_PRESENT
...:
                Case cSACL
                    objSD.ControlFlags = objSD.ControlFlags And _
                                         (NOT SE_SACL_PRESENT)
        End Select
...:
...:
...:
                End If

                boolRemoveAce = True
            Else

                strDomainName = ExtractUserDomain (strTrustee)
                strUserName = ExtractUserID (strTrustee)

                intIndice2 = 0
                For intIndice1 = 0 To UBound(arrayACL)
                    Set objACE = arrayACL(intIndice1)
                    Set objTrustee = objACE.Trustee
                    If Ucase(objTrustee.Name) = Ucase(strUserName) Then
                       If (objACE.AceFlags And INHERITED_ACE) = INHERITED_ACE Then
                          Wscript.Echo "Trustee '" & strTrustee & _
                                       "' is inherited and therefore can't be removed."

                          ReDim Preserve arrayNewACL(intIndice2)
                          Set arrayNewACL(intIndice2) = objACE
                          intIndice2 = intIndice2 + 1
                       Else
                          Wscript.Echo "Trustee '" & strTrustee & _
                                       "' has been removed from security descriptor."

                          If UBound(arrayACL) = 0 Then
...:
...:
...:
        Select Case intACLType
               Case cDACL
                    Set objNewACE = objWMIServices.Get("Win32_ACE").SpawnInstance_()

                    objNewACE.Trustee = CreateTrustee ("Everyone",
                                                       strSIDResolutionDC, _
                                                       strUserID, _
                                                       strPassword)
                    objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
                    objNewACE.AccessMask = ADS_RIGHT_GENERIC_ALL
                    objNewACE.AceFlags = CONTAINER_INHERIT_ACE Or _
                                         OBJECT_INHERIT_ACE

                    Wscript.Echo "All ACE removed,Trustee 'Everyone'" & _
```

```
224:                                    "(Full Control) has been created."
225:
226:                          ReDim arrayNewACL (0)
227:                          Set arrayNewACL (0) = objNewACE
228:
229:                          objSD.ControlFlags = objSD.ControlFlags Or _
230:                                               SE_DACL_PRESENT
...:
233:                     Case cSACL
234:                          objSD.ControlFlags = objSD.ControlFlags And _
235:                                               (NOT SE_SACL_PRESENT)
236:             End Select
...:
...:
...:
237:                               End If
238:
239:                               boolRemoveAce = True
240:                            End If
241:                          Else
242:                            ReDim Preserve arrayNewACL(intIndice2)
243:                            Set arrayNewACL(intIndice2) = objACE
244:                            intIndice2 = intIndice2 + 1
245:                          End If
...:
249:                       Next
250:                    End If
251:
252:                    intAceCount = Ubound(arrayNewACL)
253:                    If Err.Number = 9 Then
254:                       Err.Clear
255:
256:                       Wscript.Echo "All ACE removed."
257:
258:                       Select Case intACLType
259:                              Case cDACL
260:                                   objSD.DACL = Null
261:                              Case cSACL
262:                                   objSD.SACL = Null
263:                       End Select
264:                    Else
265:                       Select Case intACLType
266:                              Case cDACL
267:                                   objSD.DACL = arrayNewACL
268:                              Case cSACL
269:                                   objSD.SACL = arrayNewACL
270:                       End Select
271:                    End If
272:                 Else
273:                    WScript.Echo "No existing ACE to remove."
274:                    WScript.Quit(1)
275:                 End If
276:
277:' Here we can't retrieve a security descriptor via this access method.
278:           Case cRegistryViaWMI, _
279:                cWMINameSpaceViaADSI
280:
281:    End Select
282:
283:    If boolRemoveAce = False Then
```

```
284:       WScript.Echo "WARNING: Implicit ACE for Trustee '" & strTrustee & _
285:                    "' NOT found in security descriptor."
286:       WScript.Quit(1)
287:    End If
288:
289:    Set DelACE = objSD
290:
291:End Function
```

Although the overall logic used in the WMI object model is the same as the logic used in the ADSI object model, in the coding details there are some important differences. From a global standpoint, we still have two parts with the same purpose as for the ADSI object model:

1. Removing all ACEs (lines 156 through 187)
2. Removing a specific ACE based on the Trustee name (lines 188 through 250)

Because the ACL in the WMI object model is represented by an array, instead of manipulating the original array content another array is populated with the ACEs that are not removed. When all ACEs are removed in a DACL, this new array is initialized with the default ACE based on the "Everyone" trustee (lines 160 through 180). When removing a specific ACE, the inherited ACEs are not removed and are copied to the new array (lines 198 through 204). The same applies for ACEs that do not match the **Trustee** DelACE() function input parameter (lines 242 through 244). Basically, when an ACE is removed from the original ACL, it is not copied to the new array. Before completion, the script checks if the new array is initialized with some ACEs (line 252). If there is no ACE, the UBound() function returns an error (line 253). In such a case, the DACL or the SACL (based on the **ACLType** parameter) is set to Null (lines 258 through 270). If the new array is initialized with one or more ACEs, no error is returned and the DACL or SACL property is assigned with this new array (lines 265 through 270).

4.12.6 Reordering ACEs

When ACEs are added or removed from an ACL, it is quite important to properly reorder ACEs, because, in some cases, adding an ACE at the top of an ACL could create undesired security access. This ordering is also called the canonical order. When editing permissions with the ACL editor (i.e., from the Windows Explorer), it is important to note that the editor saves the updated rights in canonical order. Therefore, to avoid any security trou-

bles, it is important to perform the same ordering from the **WMIManageSD.Wsf** script. The ACEs in an ACL must be sorted into these five groups:

- Access-denied ACEs, which apply to the object itself
- Access-denied ACEs, which apply to a child of the object, such as a property set or property
- Access-allowed ACEs, which apply to the object itself
- Access-allowed ACEs, which apply to a child of the object, such as a property set or property
- All inherited ACEs

The ADSI and WMI object models do not support a method to properly order ACEs in an ACL. This implies that a customized logic must handle the ACE ordering. Sample 4.49 implements the logic in the ADSI object model, while Sample 4.50 implements the logic in the WMI object model. Both samples uses a logic inspired from Microsoft Knowledge Base article Q269159.

4.12.6.1 Reordering ACEs in the ADSI object model

The overall idea of the logic is to store ACEs in several temporary ACLs (one per category). Once completed, these temporary ACLs are read in the requested order and each ACE is stored in a new ACL. The end result is that this new ACL contains all ACEs in the required order.

Sample 4.49 *Reordering ACE in the ADSI object model (Part I)*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function ReOrderACE(objWMIServices, objSD, intSDType)
..:
39:    Select Case intSDType
40:' Here we have an ADSI security descriptor representation
41:          Case cFileViaADSI, _
42:               cShareViaADSI, _
43:               cActiveDirectoryViaWMI, cActiveDirectoryViaADSI, _
44:               cExchange2000MailboxViaWMI, cExchange2000MailboxViaADSI, _
45:               cExchange2000MailboxViaCDOEXM, _
46:               cRegistryViaADSI, _
47:               cWMINameSpaceViaWMI
48:
49:               ' Only the DACL is re-ordered
50:               Set objACL = objSD.DiscretionaryAcl
51:
52:               If objACL.AceCount Then
53:                  Set objNewACL = CreateObject("AccessControlList")
```

```
54:                 Set objImplicitDenyACL = CreateObject("AccessControlList")
55:                 Set objImplicitDenyObjectACL = CreateObject("AccessControlList")
56:                 Set objImplicitAllowACL = CreateObject("AccessControlList")
57:                 Set objImplicitAllowObjectACL = CreateObject("AccessControlList")
58:                 Set objInheritedACL = CreateObject("AccessControlList")
59:
60:                 For Each objACE In objACL
61:                     If ((objACE.AceFlags And ADS_ACEFLAG_INHERITED_ACE) = _
62:                                              ADS_ACEFLAG_INHERITED_ACE) Or _
63:                        ((objACE.AceFlags And INHERITED_ACE) = INHERITED_ACE) Then
64:                        objInheritedACL.AddAce objACE
65:                     Else
66:                        Select Case objACE.AceType
67:                               Case ADS_ACETYPE_ACCESS_DENIED,ACCESS_DENIED_ACE_TYPE
68:                                    objImplicitDenyACL.AddAce objACE
69:                               Case ADS_ACETYPE_ACCESS_DENIED_OBJECT
70:                                    objImplicitDenyObjectACL.AddAce objACE
71:                               Case ADS_ACETYPE_ACCESS_ALLOWED,ACCESS_ALLOWED_ACE_TYPE
72:                                    objImplicitAllowACL.AddAce objACE
73:                               Case ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
74:                                    objImplicitAllowObjectACL.AddAce objACE
75:                               Case Else
76:                                    ' Bad objACE
77:                        End Select
78:                     End If
79:                 Next
80:
81:                 ' Implicit Deny ACE.
82:                 For Each objACE In objImplicitDenyACL
83:                     objNewACL.AddAce objACE
84:                 Next
85:
86:                 ' Implicit Deny ACE Objects.
87:                 For Each objACE In objImplicitDenyObjectACL
88:                     objNewACL.AddAce objACE
89:                 Next
90:
91:                 ' Implicit Allow ACE.
92:                 For Each objACE In objImplicitAllowACL
93:                     objNewACL.AddAce objACE
94:                 Next
95:
96:                 ' Implicit Allow ACE Objects.
97:                 For Each objACE In objImplicitAllowObjectACL
98:                     objNewACL.AddAce objACE
99:                 Next
100:
101:                ' Inherited ACE.
102:                For Each objACE In objInheritedACL
103:                    objNewACL.AddAce objACE
104:                Next
105:
106:                objNewACL.AclRevision = objACL.AclRevision
107:
108:                objSD.DiscretionaryAcl = objNewACL
...:
118:            End If
119:
...:
...:
...:
```

At line 50, the ReOrderACE() function extracts the ACL from the DACL. Note that the SACL is not considered, since the ACE order is only important for security accesses. Next, if ACEs are contained in the DACL (line 52), the script creates five new ACL objects: one for the new ordered ACL (line 53) and one new ACL for each category (lines 54 through 58). For each ACE in the ACL, the script enumerates all ACEs and stores each of them in its corresponding ACL category based on its type (lines 60 through 79). Once complete, the new ACL is built from each ACL category created (lines 81 through 104). The end result is a list of ACEs properly ordered. Before returning, the new ACL is stored back to the DACL (line 108).

4.12.6.2 Reordering ACEs in the WMI object model

The ACE reordering under the WMI object model is the same as the ACE ordering in the ADSI object model. Of course, since ACLs under WMI are represented by an array, the code manipulates items in different arrays (one per category). Except for this difference, the logic of Sample 4.50 is exactly the same as Sample 4.49.

Sample 4.50 *Reordering ACE in the WMI object model (Part II)*

```
...:
...:
...:
119:
120:' Here we have a WMI security descriptor representation
121:          Case cFileViaWMI, _
122:               cShareViaWMI
123:
124:               ' Only the DACL is re-ordered
125:               arrayACL = objSD.DACL
126:
127:               If IsArray (arrayACL) Then
128:                  For intIndice = 0 To UBound(arrayACL)
129:                      If ((arrayACL(intIndice).AceFlags And ADS_ACEFLAG_INHERITED_ACE) = _
130:                                                            ADS_ACEFLAG_INHERITED_ACE) Or _
131:                         ((arrayACL(intIndice).AceFlags And INHERITED_ACE)=INHERITED_ACE) Then
132:                         intInheritedACL = intInheritedACL + 1
133:                         ReDim Preserve arrayInheritedACL(intInheritedACL)
134:                         Set arrayInheritedACL(intInheritedACL) = arrayACL(intIndice)
135:                      Else
...:
136: Select Case arrayACL(intIndice).AceType
137:        Case ADS_ACETYPE_ACCESS_DENIED,ACCESS_DENIED_ACE_TYPE
138:             intImplicitDenyACL = intImplicitDenyACL + 1
139:             ReDim Preserve arrayImplicitDenyACL(intImplicitDenyACL)
140:             Set arrayImplicitDenyACL(intImplicitDenyACL) = arrayACL(intIndice)
141:        Case ADS_ACETYPE_ACCESS_DENIED_OBJECT
142:             intImplicitDenyObjectACL = intImplicitDenyObjectACL + 1
```

```
143:             ReDim Preserve arrayImplicitDenyObjectACL(intImplicitDenyObjectACL)
144:             Set arrayImplicitDenyObjectACL(intImplicitDenyObjectACL) = arrayACL(intIndice)
145:        Case ADS_ACETYPE_ACCESS_ALLOWED,ACCESS_ALLOWED_ACE_TYPE
146:             intImplicitAllowACL = intImplicitAllowACL + 1
147:             ReDim Preserve arrayImplicitAllowACL(intImplicitAllowACL)
148:             Set arrayImplicitAllowACL(intImplicitAllowACL) = arrayACL(intIndice)
149:        Case ADS_ACETYPE_ACCESS_ALLOWED_OBJECT
150:             intImplicitAllowObjectACL = intImplicitAllowObjectACL + 1
151:             ReDim Preserve arrayImplicitAllowObjectACL(intImplicitAllowObjectACL)
152:             Set arrayImplicitAllowObjectACL(intImplicitAllowObjectACL) = arrayACL(intIndice)
153:        Case Else
154:             ' Bad ACE
155: End Select
...:
156:                     End If
157:                 Next
158:
159:                 ' Implicit Deny ACE.
160:                 If intImplicitDenyACL Then
161:                    For intIndice = 1 To Ubound(arrayImplicitDenyACL)
162:                        ReDim Preserve arrayNewACL(intNewACL)
163:                        Set arrayNewACL(intNewACL) = arrayImplicitDenyACL(intIndice)
164:                        intNewACL = intNewACL + 1
165:                    Next
166:                 End If
167:
168:                 ' Implicit Deny ACE Objects.
169:                 If intImplicitDenyObjectACL Then
170:                    For intIndice = 1 To Ubound(arrayImplicitDenyObjectACL)
171:                        ReDim Preserve arrayNewACL(intNewACL)
172:                        Set arrayNewACL(intNewACL) = arrayImplicitDenyObjectACL(intIndice)
173:                        intNewACL = intNewACL + 1
174:                    Next
175:                 End If
176:
177:                 ' Implicit Allow ACE.
178:                 If intImplicitAllowACL Then
179:                    For intIndice = 1 To Ubound(arrayImplicitAllowACL)
180:                        ReDim Preserve arrayNewACL(intNewACL)
181:                        Set arrayNewACL(intNewACL) = arrayImplicitAllowACL(intIndice)
182:                        intNewACL = intNewACL + 1
183:                    Next
184:                 End If
185:
186:                 If intImplicitAllowObjectACL Then
187:                    ' Implicit Allow ACE objects.
188:                    For intIndice = 1 To Ubound(arrayImplicitAllowObjectACL)
189:                        ReDim Preserve arrayNewACL(intNewACL)
190:                        Set arrayNewACL(intNewACL) = arrayImplicitAllowObjectACL(intIndice)
191:                        intNewACL = intNewACL + 1
192:                    Next
193:                 End If
194:
195:                 If intInheritedACL Then
196:                    ' Inherited ACE.
197:                    For intIndice = 1 To Ubound(arrayInheritedACL)
198:                        ReDim Preserve arrayNewACL(intNewACL)
199:                        Set arrayNewACL(intNewACL) = arrayInheritedACL(intIndice)
200:                        intNewACL = intNewACL + 1
201:                    Next
```

```
202:                    End If
203:
204:                    objSD.DACL = arrayNewACL
205:                End If
206:
207:' Here we do not have a security descriptor available via this access method.
208:           Case cRegistryViaWMI, _
209:                cWMINameSpaceViaADSI
210:
211:    End Select
212:
213:    Set ReOrderACE = objSD
214:
215:End Function
```

4.13 Updating the security descriptor

Now that the security descriptors can be read from protected entities via WMI or ADSI, their components, such as the *Control Flags*, can be updated; ACEs can be added and removed from DACL or SACL, but security descriptors must be saved back to the protected entities to make the security change effective. Because the security descriptor access methods differ if they come from a file, Active Directory, or a CIM repository namespace, the object model used to save security descriptors will be the same as the one used to read the security descriptor. Of course, even if the object model remains the same to save the security descriptor, the invoked method will be different. In the next sections, we will see how the security descriptor can be saved back to a protected entity in regard to its origin and the object model used.

4.13.1 Updating file and folder security descriptors

In section 4.7.1 ("Retrieving file and folder security descriptor"), both object models offered an access method to read a security descriptor. To save the security descriptor back to the file system, both object models will be considered in the same way.

4.13.1.1 Updating file and folder security descriptors with WMI

In Chapter 2, when we worked with the *CIM_LogicalFile* class supported by the *Win32* provider, although we didn't use all methods, we discussed the methods exposed by this class (see Table 2.17, "The *CIM_LogicalFile* Methods") to change the security permissions. We had:

- The *ChangeSecurityPermissions* or *ChangeSecurityPermissionsEx* methods: These methods change the security permissions for the logical file specified in the object path. If the logical file is a folder, then

ChangeSecurityPermissions will act recursively, changing the security permissions of all files and subfolders the folder contains.

- The *TakeOwnerShip* or *TakeOwnerShipEx* methods: These methods obtain ownership of the logical file specified in the object path. If the logical file is actually a directory, then *TakeOwnerShip* acts recursively, taking ownership of all the files and subdirectories the directory contains. We already demonstrated the *TakeOwnerShipEx* method in Chapter 2, with Sample 2.36.

On the other hand, the *Win32_LogicalFileSecuritySetting* class, supported by the *Security* provider, exposes two methods to retrieve and update the security descriptor:

- *GetSecurityDescriptor* method: Retrieves a structural representation of the object's security descriptor.
- *SetSecurityDescriptor* method: Replaces a structural representation of the object's security descriptor.

Although the methods from the *Win32_LogicalFileSecuritySetting* and *CIM_LogicalFile* classes seem to be redundant, they are not! The *CIM_LogicalFile ChangeSecurityPermissions* method is able to change one or several security descriptor subcomponents (owner, group, Discretionary ACL, or System ACL) recursively in a File System directory tree. On the other hand, the *Win32_LogicalFileSecuritySetting SetSecurityDescriptor* method replaces the complete structural representation of a security descriptor of a file or a directory object represented by a *Win32_LogicalFileSecuritySetting* instance (see Sample 4.51).

Sample 4.51 *Updating file and folder security descriptors with WMI (Part I)*

```
 .:
 .:
 .:
 8:' ------------------------------------------------------------------------------------------
 9:Function SetSecurityDescriptor (objConnection, objSD, strSource, intSDType)
..:
21:    SetSecurityDescriptor = False
22:
23:    Select Case intSDType
24:' +-------------------------------------------------------------------------------------+
25:' | File or Folder                                                                      |
26:' +-------------------------------------------------------------------------------------+
27:          Case cFileViaWMI
28:' WMI update technique ---------------------------------------------------------------------
29:
30:' Here objSD contains a security descriptor in the WMI object model.
31:
```

```
32:             WScript.Echo "Setting File or Folder security descriptor via WMI to '" & _
33:                          strSource & "'."
34:
35:             ' Set objWMIInstance = objConnection.Get("CIM_LogicalFile='" & strSource & "'")
..:
38:             ' intRC = objWMIInstance.ChangeSecurityPermissionsEx (objSD, _
39:             '                                                     13, _
40:             '                                                     Null, _
41:             '                                                     Null, _
42:             '                                                     True)
43:
44:             Set objWMIInstance = objConnection.Get("Win32_LogicalFileSecuritySetting='" &
45:                                  strSource & "'")
..:
48:             intRC = objWMIInstance.SetSecurityDescriptor (objSD)
49:
50:             If intRC Then
51:                WScript.Echo vbCRLF & "Failed to set File or Folder security " & _
52:                             "descriptor to '" & strSource & "' (" & intRC & ")."
53:             Else
54:                SetSecurityDescriptor = True
55:             End If
..:
..:
..:
```

By default, the code replaces the structural representation of the security descriptor (lines 44 through 48). However, the commented out lines 35 through 42 show the coding technique to use if you do want to make a recursive replacement with the *ChangeSecurityPermissionsEx* method of the *CIM_LogicalFile* class. Note that the *ChangeSecurityPermissionsEx* method does not necessarily replace the whole security descriptor. A set of flags, listed in Table 4.26, determines which part of the security descriptor needs to be replaced.

Nothing forces you to use the *ChangeSecurityPermissionsEx* method. By default, the script makes use of the *SetSecurityDescriptor* method exposed by the *Win32_LogicalFileSecuritySetting* class. Eventually, the script can process the recursion itself. Of course, this logic must be developed in the script code, but the Microsoft Knowledge Base article Q266461 shows an example for the File System security descriptor. In its current version, the SetSe-

Table 4.26 *The ChangeSecurityPermissionsEx Method Flags for the Security Descriptor Recursive Update*

ChangeSecurityPermissionsEx flags	Bit	Value
Change_Owner_Security_Information	0	1
Change_Group_Security_Information	1	2
Change_Dacl_Security_Information	2	4
Change_Sacl_Security_Information	3	8

curityDescriptor() function does not support the recursive application of a security descriptor, but the script can easily be expanded.

4.13.1.2 Updating file and folder security descriptors with ADSI

Through ADSI, under Windows Server 2003 (or Windows XP), the script makes use of the **IADsSecurityUtility** interface. This interface was already used in section 4.7.1.2 ("Retrieving file and folder security descriptors with ADSI") and follows the same logic to save the security descriptor back to the File System. Of course, a different method must be used, as shown in Sample 4.52 (lines 73 through 76). Under Windows 2000 (or Windows NT 4.0), the **ADsSecurity** object is used to save the security descriptor (line 79).

The ADSI techniques do not expose functionality to recursively save the security descriptor, although WMI does with the *ChangeSecurityPermissions-Ex* method. To accomplish this, it is necessary to recursively browse the File System hierarchy, as described in the Microsoft Knowledge Base article Q266461.

Sample 4.52 *Updating file and folder security descriptors with ADSI (Part II)*

```
..:
..:
..:
58:
59:           Case cFileViaADSI
60:' ADSI update technique ------------------------------------------------------------------------
61:
62:' Here objSD contains a security descriptor in the ADSI object model.
63:
64:               WScript.Echo "Setting File or Folder security descriptor via ADSI to '" & _
65:                            strSource & "'."
66:
67:               ' Windows Server 2003 only ------------------------------------------------
68:               objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
69:                                             ADS_SECURITY_INFO_GROUP Or _
70:                                             ADS_SECURITY_INFO_DACL ' Or _
71:                                             ' ADS_SECURITY_INFO_SACL
72:
73:               objADsSecurity.SetSecurityDescriptor strSource, _
74:                                                    ADS_PATH_FILE, _
75:                                                    objSD, _
76:                                                    ADS_SD_FORMAT_IID
77:
78:               ' Windows 2000 only -------------------------------------------------------
79:               ' objADsSecurity.SetSecurityDescriptor objSD, "FILE://" & strSource
80:
81:               If Err.Number Then
82:                  Err.Clear
83:                  WScript.Echo vbCRLF & "Failed to set File or Folder security " & _
84:                               "descriptor to '" & strSource & "' (" & intRC & ")."
```

```
85:                Else
86:                   SetSecurityDescriptor = True
87:                End If
88:
..:
..:
..:
```

Since the **ADsSecurityUtility** object is subject to a bug under Windows Server 2003 and Windows XP, don't forget to refer to section 4.7.1.2 ("Retrieving file and folder security descriptors with ADSI"). If the work-around explained in that section is not used, then it will be impossible to save the updated SACL security descriptor back to the file system.

4.13.2 Updating File System share security descriptors

When a File System share security descriptor is read (see section 4.7.2, "Retrieving file system share security descriptor"), it is not always certain that a security descriptor is available from the share. In such a case, when no security descriptor is available, a default security descriptor is created based on the "Everyone" trustee. To save the security descriptor back to the File System share, we will see that some care must be taken.

4.13.2.1 Updating share security descriptors with WMI

With the WMI object model it is necessary to retrieve an instance of the File System share in an **SWBemObject** to update the security descriptor (see Sample 4.53).

Sample 4.53 *Updating share security descriptors with WMI (Part III)*

```
..:
..:
..:
88:
89:' +-------------------------------------------------------------------------------------------+
90:' | Share                                                                                     |
91:' +-------------------------------------------------------------------------------------------+
92:          Case cShareViaWMI
93:' WMI update technique ------------------------------------------------------------------------
94:
95:' Here objSD contains a security descriptor in the WMI object model.
96:
97:                WScript.Echo "Setting Share security descriptor " & _
98:                             "via WMI to '" & strSource & "'."
99:
100:               Set objWMIInstance = objConnection.Get("Win32_LogicalShareSecuritySetting='" &
101:                                                      strSource & "'")
102:               If Err.Number = wbemErrNotFound Then
103:                  Err.Clear
104:                  Set objWMIInstance = objConnection.Get("Win32_Share='" & strSource & "'")
```

```
...:
107:              intRC = objWMIInstance.SetShareInfo (0, "", objSD)
108:              If intRC Then
109:                 WScript.Echo vbCRLF & "Failed to set Share security descriptor to '" & _
110:                              strSource & "' (" & intRC & ")."
111:              Else
112:                 SetSecurityDescriptor = True
113:              End If
114:           Else
115:              If Err.Number Then ErrorHandler (Err)
116:
117:              intRC = objWMIInstance.SetSecurityDescriptor (objSD)
118:              If intRC Then
119:                 WScript.Echo vbCRLF & "Failed to set Share security descriptor to '" & _
120:                              strSource & "' (" & intRC & ")."
121:              Else
122:                 SetSecurityDescriptor = True
123:              End If
124:           End If
...:
...:
...:
```

The *Win32_LogicalShareSecuritySetting* class can be used to retrieve an instance of the File System share but with the condition that the share has a security descriptor set (lines 100 and 101). If no security descriptor exists, a **wbemErrNotFound** error will be returned, even if the share is well defined (line 102). In such a case the *SetSecurityDescriptor* method of the *Win32_LogicalShareSecuritySetting* class can't be used to save the security descriptor. At that time, the script must retrieve an instance of the File System share by using another class: the *Win32_Share* class (line 104) discussed in Chapter 2 with Sample 2.65. This class exposes the *SetShareInfo* method (line 107). The third input parameter of this method requires a structural WMI representation of the security descriptor (made from the *Win32_SecurityDescriptor* class). This technique saves to the File System share the updated or created security descriptor (see Sample 4.25, "Create a default security descriptor for a share").

4.13.2.2 Updating share security descriptors with ADSI

To update the File System share security descriptor with ADSI, it is mandatory to run under Windows Server 2003 (or Windows XP) and make use of the **IADsSecurityUtility** interface, as shown in Sample 4.54. Under Windows 2000 (or Windows NT 4.0), there is no way to update a File System share security descriptor with ADSI.

However, things are easier with ADSI than they are with WMI, since it is not necessary in the ADSI object model to retrieve an instance of the share to update the security descriptor (lines 137 through 140). Only the

SetSecurityDescriptor method of the **IADsSecurityUtility** interface needs to be used.

Sample 4.54 *Updating share security descriptors with ADSI (Part IV)*

```
...:
...:
...:
127:
128:          Case cShareViaADSI
129:' ADSI update technique ------------------------------------------------------------------
130:
131:' Here objSD contains a security descriptor in the ADSI object model.
132:
133:              WScript.Echo "Setting Share security descriptor via ADSI to '" & _
134:                           strSource & "'."
135:
136:              ' Windows Server 2003 only ---------------------------------------------
137:              objADsSecurity.SetSecurityDescriptor strSource, _
138:                                                   ADS_PATH_FILESHARE, _
139:                                                   objSD, _
140:                                                   ADS_SD_FORMAT_IID
141:
142:              If Err.Number Then
143:                 Err.Clear
144:                    WScript.Echo vbCRLF & "Failed to set Share security descriptor to '" & _
145:                                 strSource & "' (" & intRC & ")."
146:              Else
147:                 SetSecurityDescriptor = True
148:              End If
149:
...:
...:
...:
```

4.13.3 Updating Active Directory object security descriptors

In section 4.7.3 ("Retrieving Active Directory object security descriptors"), the script retrieves the security descriptor via WMI and ADSI. To update the security descriptor back to Active Directory the script will also use the WMI and ADSI techniques.

4.13.3.1 Updating Active Directory object security descriptors with WMI

Updating the security descriptor in Active Directory is little bit more complex than the previous security descriptor update mechanisms. This logic is implemented in Sample 4.55. First, the script must retrieve an instance of the Active Directory object secured by the security descriptor to update (lines 158 through 165). As with the security descriptor access in Sample 4.18 ("Retrieving Active Directory object security descriptors with WMI

[Part V]"), the script invokes the ConvertStringInArray() function to split the Active Directory object class from the object **distinguishedName** (line 158).

Sample 4.55 *Updating Active Directory object security descriptors with WMI (Part V)*

```
...:
...:
...:
149:
150:' +-------------------------------------------------------------------------------------------+
151:' | Active Directory object                                                                   |
152:' +-------------------------------------------------------------------------------------------+
153:          Case cActiveDirectoryViaWMI
154:' WMI update technique ------------------------------------------------------------------------
155:
156:' Here objSD contains a security descriptor in the ADSI object model.
157:
158:               arrayADInfo = ConvertStringInArray (strSource, ";")
159:               WScript.Echo "Setting " & LCase(arrayADInfo(0)) & _
160:                            " Active Directory object security descriptor " & _
161:                            "via WMI to 'LDAP://" & arrayADInfo(1) & "'."
162:
163:               Set objWMIInstance = objConnection.Get("ds_" & LCase(arrayADInfo(0)) & _
164:                                                      ".ADSIPath='LDAP://" & _
165:                                                      arrayADInfo(1) & "'")
...:
168:               arrayBytes = objADSIHelper.ConvertAdsiSDToRawSD (objSD)
...:
171:               Set objTempSD = objWMIInstance.DS_nTSecurityDescriptor
...:
174:               objTempSD.Value = arrayBytes
175:               Set objWMIInstance.DS_nTSecurityDescriptor = objTempSD
176:
177:               objWMINamedValueSet.Add "__PUT_EXTENSIONS", True
178:               objWMINamedValueSet.Add "__PUT_EXT_CLIENT_REQUEST", True
179:               objWMINamedValueSet.Add "__PUT_EXT_PROPERTIES", _
180:                                       Array ("DS_nTSecurityDescriptor")
181:
182:               objWMIInstance.Put_ wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete, _
183:                                   objWMINamedValueSet
184:               If Err.Number Then
185:                  Err.Clear
186:                  WScript.Echo vbCRLF & "Failed to set Active Directory object " & _
187:                               "security descriptor to 'LDAP://" & strSource & "'."
188:               Else
189:                  SetSecurityDescriptor = True
190:               End If
...:
...:
...:
```

Because the WMI security descriptor method retrieves a security descriptor in a raw form, the WMI security descriptor update method must set the security descriptor in the same format. This requires a conversion of

the security descriptor from its ADSI form to a binary form. This conversion is made with the help of the ActiveX DLL (line 168) used during the read operation of the security descriptor. During the read operation of the security descriptor (see Sample 4.18), we saw that the security descriptor is contained in an **SWBemNamedValue** object assigned to the *DS_nTSecurityDescriptor* property of the **SWBemObject** representing the Active Directory object instance. Because the script saves the updated security descriptor back to the Active Directory, it sets the value of the **SWBemNamedValue** object (line 174) to the *DS_nTSecurityDescriptor* property (line 175). Now that the security descriptor of the **SWBemObject** object representing the Active Directory object is updated, this WMI instance must be updated back to the Active Directory. Here, the script uses a WMI partial update technique (lines 177 through 183). We already used the partial-instance update technique in Chapter 3 (section 3.6.1.1, "Creating and updating objects in Active Directory"). Actually, it is impossible to save the complete Active Directory object instance at once, because some of the Active Directory attributes defined in this instance can only be set by the system itself. This means that if we try to save the complete instance back to Active Directory without a partial-instance update, Active Directory will reject the update, since it will consider this update as a violation of the rules enforced by the directory schema (constraint violation). Therefore, the script uses the partial-instance update mechanism and specifies that only the **ntSecurityDescriptor** Active Directory attribute (lines 179 through 180), represented by the *DS_nTSecurityDescriptor* WMI property, must be updated.

4.13.3.2 Updating Active Directory object security descriptors with ADSI

Updating an Active Directory security descriptor with ADSI is much simpler than the WMI technique (see Sample 4.56). To ensure that all security descriptor components are saved (i.e., DACL, SACL), the *SetOption* property is initialized accordingly (lines 201 through 204). Next, the security descriptor is placed back into the **nTSecurityDescriptor** attribute (line 206) and committed back to the Active Directory (line 209).

Sample 4.56 *Updating Active Directory object security descriptors with ADSI (Part VI)*

```
...:
...:
...:
193:
194:          Case cActiveDirectoryViaADSI
195:' ADSI update technique ------------------------------------------------------------------------
196:
197:' Here objSD contains a security descriptor in the ADSI object model.
198:
```

```
199:              WScript.Echo "Setting Active Directory object security " & _
200:                           "descriptor via ADSI to 'LDAP://" & strSource & "'."
201:              objConnection.SetOption ADS_OPTION_SECURITY_MASK, ADS_SECURITY_INFO_OWNER Or _
202:                                                                ADS_SECURITY_INFO_GROUP Or _
203:                                                                ADS_SECURITY_INFO_DACL Or _
204:                                                                ADS_SECURITY_INFO_SACL
205:
206:              objConnection.Put "ntSecurityDescriptor", objSD
...:
209:              objConnection.SetInfo
210:              If Err.Number Then
211:                 Err.Clear
212:                 WScript.Echo vbCRLF & "Failed to set Active Directory object " & _
213:                              "security descriptor to 'LDAP://" & strSource & "'."
214:              Else
215:                 SetSecurityDescriptor = True
216:              End If
217:
...:
...:
...:
```

4.13.4 Updating Exchange 2000 mailbox security descriptors

Updating the Exchange 2000 mailbox security descriptor requires some specific considerations. With the initial release of Exchange 2000 and with Service Pack 1, the use of ADSI and CDOEXM to manage the mailbox security descriptor was quite limited. All of the information required to create an Exchange 2000 mailbox is initially stored in the Active Directory. By initializing a specific set of attributes associated with a user object, it is possible to create a mailbox using ADSI. Although not officially supported by Microsoft, this is technically possible (at least until Exchange 2000 Service Pack 3). The supported Microsoft solution is to use the CDOEXM extension to create the mailbox. Basically, CDOEXM abstracts the set of attributes required via the *CreateMailbox* method exposed by the **IMailboxStore** CDOEXM interface. This gives the guarantee of a reliable code, even if Microsoft changes the underlying logic of the mailbox creation in the future. Put simply, the ADSI technique can be seen as a raw mailbox creation process, while the CDOEXM technique can be seen as a logic encapsulated in a COM object.

Despite this nice abstraction layer, until Exchange 2000 Service Pack 2, the original CDOEXM version does not expose any attributes or methods to update mailbox security. Instead, CDOEXM configures a default security for the mailbox, which is not always the one required by Administrators. One possible solution is to use ADSI and access the **msExchMailboxSecurityDescriptor** attribute to initialize the Exchange

2000 mailbox security descriptor as required. But this technique has its limitations, since it only works well for new mailboxes, not existing mailboxes. Why? When an Exchange 2000 mailbox is created, the information required for the definition of the mailbox is stored in the Active Directory. Actually, the physical mailbox, the one in the Exchange 2000 Store, is not created immediately. This "physical" mailbox is created only when the user connects to the new mailbox or when mail is delivered to it. At that time, the mailbox is really created in the Exchange Store, and its security is set from the **msExchMailboxSecurityDescriptor** attribute in the Exchange 2000 Store.

The Active Directory **msExchMailboxSecurityDescriptor** becomes a copy of the real security descriptor set in the store and it only mirrors its value. In this case, ADSI can't be used anymore to change the mailbox security, since it does not provide access to the security descriptor in the store. Any change to the **msExchMailboxSecurityDescriptor** attribute will not be reflected in the Store. So, this solution only works for brand new mailboxes before they are "physically" created in the Exchange Store. Actually, the **msExchMailboxSecurityDescriptor** has two purposes:

- The attribute is a placeholder used to store the security information that will be written to the mailbox security descriptor in the Exchange store when the "physical" mailbox is created.
- Since the attribute is replicated in the Exchange organization, it allows the mailbox permissions to be accessible to all Exchange servers that are connected to different GCs. The evaluation of the "Send as" permission (Extended Right) requires the mailbox permissions to be accessible by all Exchange servers.

Since the Exchange 2000 Service Pack 2, it is possible to update the mailbox security descriptor in the Exchange Store using a new property, called *MailboxRights*, which is exposed by the **IMailboxStore** CDOEXM interface. Since CDOEXM acts as an ADSI extension, this property is associated with the ADSI **IADsUser** interface.

Based on this technical explanation, we can conclude that the Exchange 2000 mailbox security descriptor can be read with any access method (WMI, ADSI, CDOEXM), but it can only be updated with:

- WMI, if the mailbox is not physically created in the Exchange 2000 Store.
- ADSI, if the mailbox is not physically created in the Exchange 2000 Store.

- CDOEXM, if the mailbox is physically created or not in the Exchange 2000 Store. However, Exchange 2000 Service Pack 2 is required!

Now that the update conditions of the Exchange 2000 mailbox security descriptor are clear, let's see how this can be coded in a script.

4.13.4.1 Updating Exchange 2000 mailbox security descriptors with WMI

The Exchange 2000 mailbox security descriptor update follows the exact same logic as the update of an Active Directory object security descriptor. The only difference is in the attribute to be updated. For an Active Directory object, the script updated the *DS_nTSecurityDescriptor* of the **SWBemObject** representing the Active Directory object (see Sample 4.55). For the Exchange 2000 mailbox security descriptor, the script updates the *DS_msExchMailboxSecurityDescriptor* property of the **SWBemObject** representing the Active Directory user object (see Sample 4.57). For the rest, we use the same four steps:

- Retrieval of the Active Directory user object instance (lines 228 and 229)
- Conversion of the security descriptor to a binary format (line 232)
- Assignment of the **SWBemNamedValue** object assigned to the *DS_msExchMailboxSecurityDescriptor* property (lines 235 to 239)
- Partial update of the WMI instance representing the Active Directory user object (lines 241 through 247)

Sample 4.57 *Updating Exchange 2000 mailbox security descriptors with WMI (Part VII)*

```
...:
...:
...:
217:
218:' +-------------------------------------------------------------------------------------+
219:' | Exchange 2000 mailbox                                                               |
220:' +-------------------------------------------------------------------------------------+
221:          Case cExchange2000MailboxViaWMI
222:' WMI update technique ------------------------------------------------------------------
223:
224:' Here objSD contains a security descriptor in the ADSI object model.
225:
226:               WScript.Echo "Setting Exchange 2000 mailbox security " & _
227:                            "descriptor via WMI to 'LDAP://" & strSource & "'."
228:               Set objWMIInstance = objConnection.Get("ds_user.ADSIPath='LDAP://" & _
229:                                                      strSource & "'")
...:
232:               arrayBytes = objADSIHelper.ConvertAdsiSDToRawSD (objSD)
...:
```

```
235:                Set objTempSD = objWMIInstance.DS_msExchMailboxSecurityDescriptor
...:
238:                objTempSD.Value = arrayBytes
239:                Set objWMIInstance.DS_nTSecurityDescriptor = objTempSD
240:
241:                objWMINamedValueSet.Add "__PUT_EXTENSIONS", True
242:                objWMINamedValueSet.Add "__PUT_EXT_CLIENT_REQUEST", True
243:                objWMINamedValueSet.Add "__PUT_EXT_PROPERTIES", _
244:                                        Array ("DS_msExchMailboxSecurityDescriptor")
245:
246:                objWMIInstance.Put_ wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete, _
247:                                    objWMINamedValueSet
248:                If Err.Number Then
249:                   Err.Clear
250:                   WScript.Echo vbCRLF & "Failed to set Exchange 2000 mailbox " & _
251:                                "security descriptor to 'LDAP://" & strSource & "'."
252:                Else
253:                   SetSecurityDescriptor = True
254:                End If
...:
...:
...:
```

4.13.4.2 Updating Exchange 2000 mailbox security descriptors with ADSI

As with the ADSI update technique of a security descriptor from an Active Directory object, the update of a security descriptor of an Exchange 2000 mailbox follows the same logic and coding technique. However, instead of updating the **nTSecurityDescriptor** attribute, Sample 4.58 updates the **msExchMailboxSecurityDescriptor** attribute (lines 270 and 273).

Sample 4.58 *Updating Exchange 2000 mailbox security descriptors with ADSI (Part VIII)*

```
...:
...:
...:
257:
258:          Case cExchange2000MailboxViaADSI
259:' ADSI update technique ------------------------------------------------------------------------
260:
261:' Here objSD contains a security descriptor in the ADSI object model.
262:
263:                WScript.Echo "Setting Exchange 2000 mailbox security " & _
264:                             "descriptor via ADSI to 'LDAP://" & strSource & "'."
265:                objConnection.SetOption ADS_OPTION_SECURITY_MASK, ADS_SECURITY_INFO_OWNER Or _
266:                                                                  ADS_SECURITY_INFO_GROUP Or _
267:                                                                  ADS_SECURITY_INFO_DACL Or _
268:                                                                  ADS_SECURITY_INFO_SACL
269:
270:                objConnection.Put "msExchMailboxSecurityDescriptor", objSD
...:
273:                objConnection.SetInfo
274:                If Err.Number Then
275:                   Err.Clear
276:                   WScript.Echo vbCRLF & "Failed to set Exchange 2000 mailbox " & _
277:                                "security descriptor to 'LDAP://" & strSource & "'."
```

```
278:               Else
279:                  SetSecurityDescriptor = True
280:               End If
281:
...:
...:
...:
```

4.13.4.3 Updating Exchange 2000 mailbox security descriptors with CDOEXM

The CDOEXM mailbox security descriptor update technique is the only one able to update the security descriptor when the Exchange 2000 mailbox is physically created in the Exchange 2000 Store. As mentioned before, the Exchange 2000 Service Pack 2 is required. Basically, instead of directly updating the **msExchMailboxSecurityDescriptor** attribute (see Sample 4.58) with the line:

```
objConnection.Put "msExchMailboxSecurityDescriptor", objSD
```

the CDOEXM property, acting as an extension for ADSI, is assigned with the updated security descriptor as follows:

```
objConnection.MailboxRights = Array (objSD)
```

Behind the scenes, the logic encapsulated in the CDOEXM object will update the security descriptor in the Active Directory and at the same time in the Exchange 2000 Store. Sample 4.59 shows the complete code in the context of the script. As we can see, the code modification to support the CDOEXM is very small and the overall logic looks like the ADSI update method. This facility comes mainly from the fact that CDOEXM acts as an extension for ADSI.

Sample 4.59 *Updating Exchange 2000 mailbox security descriptors with CDOEXM (Part IX)*

```
...:
...:
...:
281:
282:          Case cExchange2000MailboxViaCDOEXM
283:' CDOEXM update technique ------------------------------------------------------------------
284:
285:' Here objSD contains a security descriptor in the ADSI object model.
286:
287:               WScript.Echo "Setting Exchange 2000 mailbox security " & _
288:                            "descriptor via CDOEXM to 'LDAP://" & strSource & "'."
289:               objConnection.MailboxRights = Array (objSD)
290:
291:               objConnection.SetInfo
292:               If Err.Number Then
293:                  Err.Clear
294:                  WScript.Echo vbCRLF & "Failed to set Exchange 2000 mailbox " & _
295:                               "security descriptor to 'LDAP://" & strSource & "'."
```

```
296:                Else
297:                   SetSecurityDescriptor = True
298:                End If
299:
...:
...:
...:
```

4.13.5 Updating registry key security descriptors with ADSI

To read a registry key security descriptor, we must determine if we are running under Windows NT 4.0, Windows 2000, Windows XP, or Windows Server 2003 (see section 4.7.5, "Retrieving registry key security descriptors with ADSI"). As we have seen, under Windows 2000 (or Windows NT 4.0), we can use the **ADsSecurity.DLL** ActiveX to read a registry security descriptor. Under Windows XP or Windows Server 2003, we should use the **IADsSecurityUtility** interface to read the security descriptor from the registry. To update the security descriptor to a registry key, the logic follows the same rule. By default, Sample 4.60 implements the Windows XP/Windows Server 2003 logic (lines 316 through 325), but the commented out lines can be swapped to run the script under Windows NT 4.0/Windows 2000 (line 328). Under Windows XP/Windows Server 2003, the *SecurityMask* is set up to ensure that all components of the security descriptor are saved back to the registry (lines 317 through 320).

Sample 4.60 *Updating registry key security descriptors with ADSI (Part X)*

```
...:
...:
...:
299:
300:' +-------------------------------------------------------------------------------------+
301:' | Registry key                                                                        |
302:' +-------------------------------------------------------------------------------------+
303:          Case cRegistryViaWMI
304:' WMI update technique ------------------------------------------------------------------
305:
306:' Here we can't retrieve a security descriptor via this access method.
307:
308:          Case cRegistryViaADSI
309:' ADSI update technique -------------------------------------------------------------------
310:
311:' Here objSD contains a security descriptor in the ADSI object model.
312:
313:               WScript.Echo "Setting registry security descriptor via ADSI to '" & _
314:                            strSource & "'."
315:
316:               ' Windows Server 2003 only ------------------------------------------------
317:               objADsSecurity.SecurityMask = ADS_SECURITY_INFO_OWNER Or _
318:                                             ADS_SECURITY_INFO_GROUP Or _
319:                                             ADS_SECURITY_INFO_DACL ' Or _
320:                                             ' ADS_SECURITY_INFO_SACL
```

```
321:
322:              objADsSecurity.SetSecurityDescriptor strSource, _
323:                                                   ADS_PATH_REGISTRY, _
324:                                                   objSD, _
325:                                                   ADS_SD_FORMAT_IID
326:
327:              ' Windows 2000 only ------------------------------------------------------
328:              ' objADsSecurity.SetSecurityDescriptor objSD, "RGY://" & strSource
329:
330:              If Err.Number Then
331:                 Err.Clear
332:                 WScript.Echo vbCRLF & "Failed to set registry security descriptor to '" & _
333:                              strSource & "'."
334:              Else
335:                 SetSecurityDescriptor = True
336:              End If
337:
...:
...:
...:
```

Since the **ADsSecurityUtility** object is subject to a bug under Windows Server 2003 and Windows XP, don't forget to refer to section 4.7.1.2 ("Retrieving file and folder security descriptors with ADSI"). If the workaround explained in that section is not used, and since the WMI technique is not able to update a structural representation of a registry key security decriptor, there will currently be no way to update the SACL of a registry key security descriptor from the scripting world with the current ADSI COM objects.

4.13.6 Updating CIM repository namespace security descriptors with WMI

When reading a CIM repository namespace security descriptor, we always retrieve the security descriptor in a raw format. Therefore, the script converts it into an ADSI structural representation (see section 4.7.6, "Retrieving CIM repository namespace security descriptors with WMI," Sample 4.24). To update the security descriptor back to a CIM repository namespace, the security descriptor must be converted back to a raw format from its ADSI structural representation. Sample 4.61 illustrates the logic.

Sample 4.61 *Updating CIM repository namespace security descriptors with WMI (Part XI)*

```
...:
...:
...:
337:
338:' +-------------------------------------------------------------------------------------+
339:' | CIM repository namespace                                                            |
340:' +-------------------------------------------------------------------------------------+
341:          Case cWMINameSpaceViaWMI
```

```
342:' WMI update technique ------------------------------------------------------------------------
343:
344:' Here objSD contains a security descriptor in the ADSI object model.
345:
346:                WScript.Echo "Setting CIM repository namespace security descriptor via WMI to '" & _
347:                             strSource & "'."
348:                Set objWMIInstance = objConnection.Get("__SystemSecurity=@")
...:
351:                arrayBytes = objADSIHelper.ConvertAdsiSDToRawSD (objSD)
...:
354:                intRC = objWMIInstance.SetSD (arrayBytes)
355:                If intRC Then
356:                   WScript.Echo vbCRLF & "Failed to set CIM repository namespace security " & _
357:                                "descriptor to '" & strSource & "'."
358:                Else
359:                   SetSecurityDescriptor = True
360:                End If
361:
362:           Case cWMINameSpaceViaADSI
363:' ADSI update technique -------------------------------------------------------------------------
364:
365:' Here we can't retrieve a security descriptor via this access method.
366:
367:           Case Else
368:
369:    End Select
370:
371:End Function
```

First, the script retrieves an instance of the *__SystemSecurity* singleton class (line 348). Next, it converts the ADSI security descriptor structural representation to a raw format (line 351) with the help of the **ADSIHelper** object (see section 4.9, "The security descriptor conversion"). Then it uses the *SetSD* method exposed by the *__SystemSecurity* class to update the security descriptor.

4.14 How WMI scripters are affected by the Microsoft security push

In early 2002, Microsoft took a huge security initiative to make the Windows platform more secure. This initiative requested all Microsoft developers to review their code for any type of potential problems that could lead to a security breach into the existing Operating Systems. This initiative implied that all new developments were put on hold for awhile until the security review of existing source codes was completed. Of course, Windows Server 2003 inherits the benefit of this security initiative, since some of its existing structures have been updated based on the discoveries. As far as WMI under Windows Server 2003 is concerned, Microsoft made some changes or enhancements at the WMI COM/DCOM level. Of course,

most of these changes concern WMI C++ programmers dealing with the WMI COM low-level interfaces. However, some of these changes impact the WMI scripting as well:

- One of the WMI COM/DCOM modifications impacts the WMI asynchronous scripting. Actually, Microsoft added one new feature under Windows Server 2003 to enforce security of the WMI asynchronous calls.
- The security descriptor set on a CIM repository namespace can't be set to NULL.
- The ADSI WMI Extension is removed under Windows Server 2003.
- A new built-in group, called the Windows Authorizations Access Group (WAAG), has been added.

4.14.1 Asynchronous scripting

To understand the impact of the security initiative on the asynchronous scripting, it is necessary to look at some of the mechanisms involved at the COM/DCOM level when WMI asynchronous operations or events are involved. However, we will not delve into the details of the COM/DCOM interfaces to explain this. Instead, we will give a high-level overview of the mechanism and how a script making use of an asynchronous call can be affected.

Generally speaking, when a WMI client application performs asynchronous operations or events, it first connects to the WMI process **WinMgmt.Exe** (see Figure 4.36, arrow 1). During this first step, the WMI client passes a sink to **WinMgmt.Exe** to allow **WinMgmt.Exe** to call the sink routine implemented by the client later on.

When working at the COM/DCOM level, the WMI client application has control of the authentication level for the **WinMgmt.Exe** outgoing calls (see the first book, *Understanding WMI Scripting*, Chapter 4, sections 4.3.3, "The security settings of the moniker," and 4.4.1, "Establishing the WMI connection"). In such a case, **WinMgmt.Exe** retrieves the WMI client security settings when called, and always attempts to call back the client sink routine with the same authentications level (see Figure 4.36, arrow 2). For obvious security reasons, it is always recommended that clients perform a security access check when the sink routine is invoked. However, when the WMI client is a WMI script or cannot control the security settings of the process or when the environment is pre-Windows 2000 (using NTLM), the machine LocalSystem does not have a network identity and it is impossible

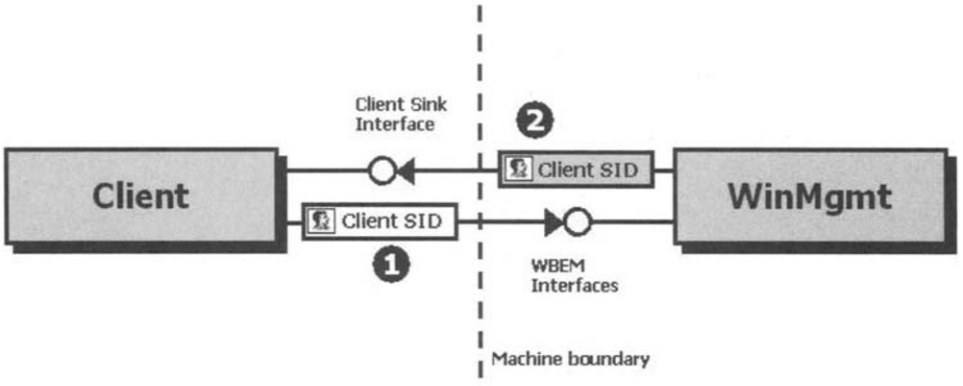


Figure 4.36 *A WMI client application performing an asynchronous operation.*

to control the security level of the callback. In such a case another action takes place, which utilizes **UnSecApp.Exe** (see Figure 4.37). Note that this slightly different action is also applicable for other applications, such as MMC snap-ins.

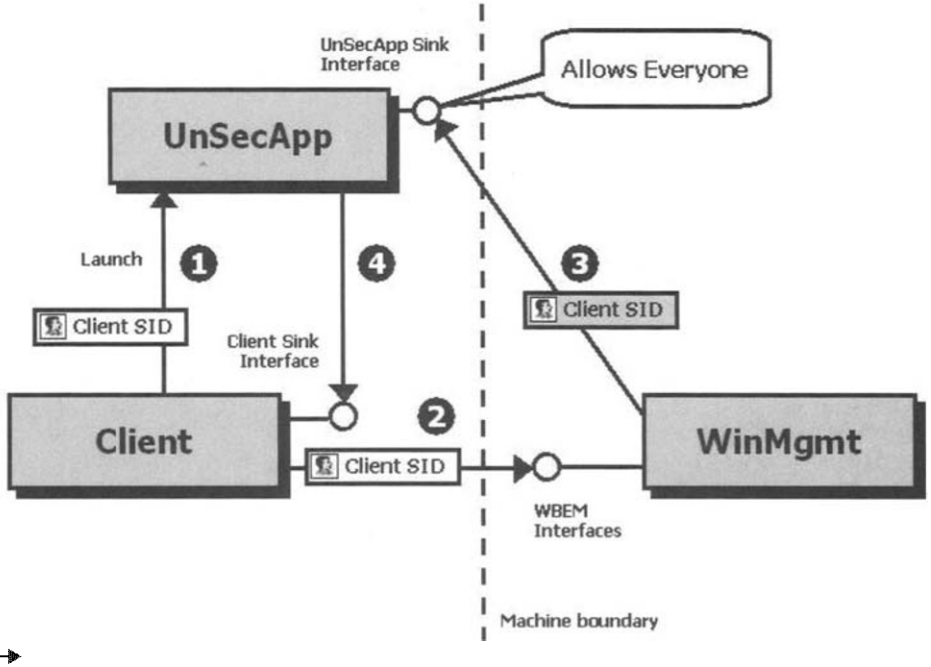


Figure 4.37 *A WMI client application performing an asynchronous operation where UnSecApp.Exe is involved.*

When a WMI script invokes an asynchronous operation, the WMI COM underlying mechanism starts **UnSecApp.Exe** in a separate process on the client (see Figure 4.37, arrow 1). Once started, **UnSecApp.Exe** passes its own sink to the client. Next, when the WMI script initiates the asynchronous call to **WinMgmt.Exe**, it passes the **UnSecApp.Exe** sink along with the call (see Figure 4.37, arrow 2). Therefore, during the **WinMgmt.Exe** callback, the sink utilized is the **UnSecApp.Exe** sink (see Figure 4.37, arrow 3). Because the WMI script can't enforce the callback security as a COM/DCOM WMI client (as shown in Figure 4.36), to get this callback working the **UnSecApp.Exe** accepts callbacks from everyone. Once the callback is accepted by **UnSecApp.Exe**, it returns the asynchronous call information back to the WMI script (see Figure 4.37, arrow 4). It is important to note that **UnSecApp.Exe** is only involved during the asynchronous operations. Therefore, synchronous and semisynchronous operations do not use the same mechanism.

When using asynchronous operations or event notifications in this implementation (Windows NT 4.0, Windows 2000, and Windows XP), it is recommended you take some precautions:

- If you can implement a logic that does not use asynchronous calls for your scripts, consider this option as the best option from a security standpoint. Therefore, you should always try to consider a synchronous or semisynchronous scripting technique when possible.
- If you must absolutely use an asynchronous scripting technique, make sure that the script does not perform some critical operations in the event sink (i.e., system shutdown) in a too powerful security context (i.e., administrative security context). If you implement an asynchronous scripting technique, it is a good idea to implement access checks in the client code as well. Obviously, as a preventive approach, it is best that you don't let scripts execute in the security context of an administrator whenever possible.
- Despite the previous recommendations, if you must perform some critical operations in the event sink in an administrative security context, as we will discuss in the next section, under Windows Server 2003 you can activate a lockdown mechanism, which Microsoft implements to secure the **WinMgmt.Exe** callbacks. Note that at writing time, there is no plan to implement a similar mechanism for the previous Operating System versions.

4.14.1.1 How the lockdown mechanism of Windows Server 2003 works

Under Windows Server 2003, Microsoft created a wrapper for **UnSecApp.Exe**. This wrapper plays the role of a security broker between the WMI client and the **WinMgmt.Exe** to validate the callbacks executed on the **UnSecApp.Exe** sink.

However, by default, this wrapper is disabled to ensure a backward compatibility with the WMI asynchronous applications previously developed. If the environment requires a stronger security, it is possible to enable the wrapper in two ways:

- **By programming:** This is only possible for the WMI COM/DCOM applications. In such case, the wrapper behavior can be determined by application, which means that one WMI client application could require authenticating the callbacks, while another will run with the default settings (Everyone).
- **By changing a registry key:** This is the technique to use for the WMI scripts, because there is no way from the scripting environment to determine the wrapper behavior. It is important to note that if the registry key is modified to enable the wrapper to only accept the authenticated calls, the setting is global for all applications (see Figure 4.38). There is no granularity available.

The registry key to change is located in the registry hive HKLM\SOFTWARE\Microsoft\WBEM\CIMOM, where the *UnsecappAccessControlDefault* value must be changed as follows:

- To allow anonymous calls (default), the *UnsecappAccessControlDefault* value must be set to 0.

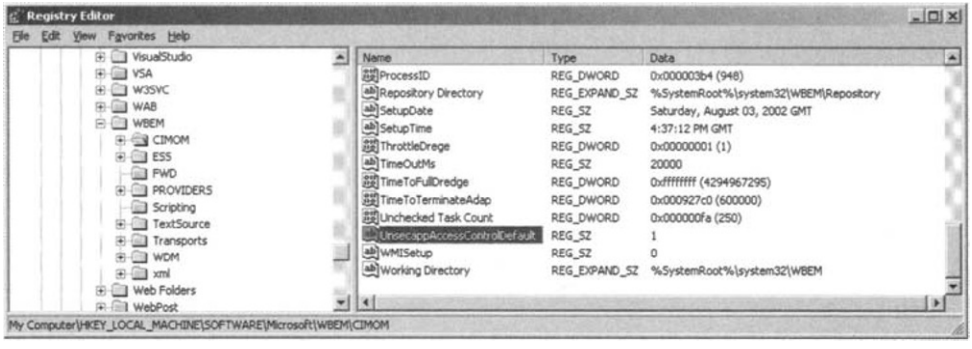


Figure 4.38 *The registry activating the new lockdown mechanism of Windows Server 2003.*

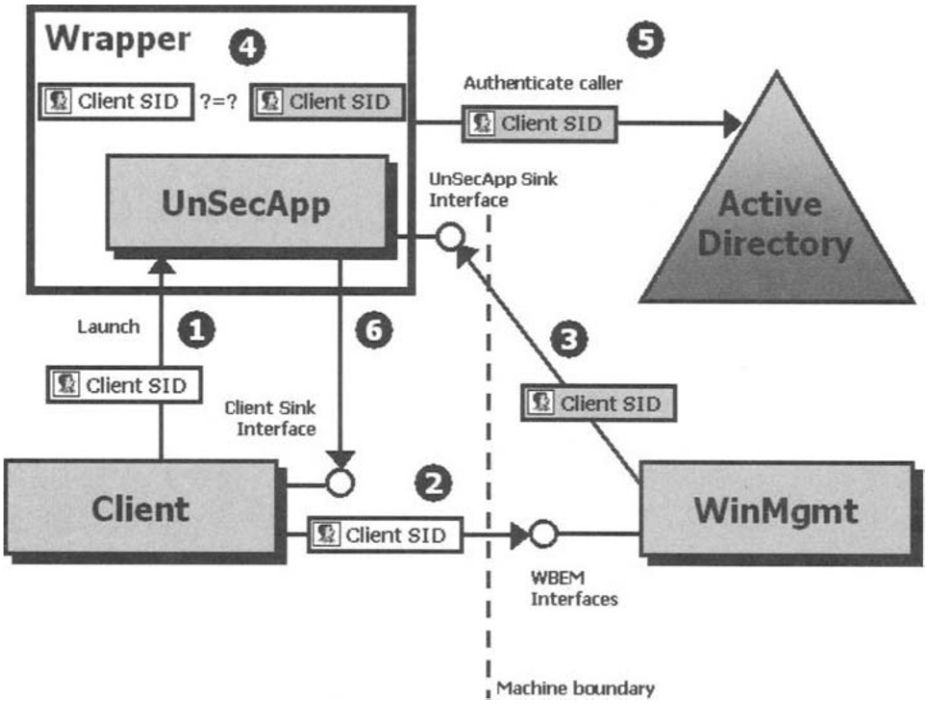


Figure 4.39 *A WMI client application performing an asynchronous operation where UnSecApp.Exe is involved and when the Windows Server 2003 lockdown mechanism is activated.*

- To allow authenticated calls (global setting), the *UnsecappAccessControlDefault* value must be set to 1. Althought the default registry key value is set to 0, it is recommended to set it to 1 if you plan to use asynchronous calls in your applications or scripts.

Once enabled, the wrapper intercepts every callback to **UnSecApp.Exe** in order to verify the identity of the component performing the callback, which should normally be **WinMgmt.Exe** (see Figure 4.39). Actually, the wrapper extracts the SID of the client application from the security token when the initial call of the client application is executed (see Figure 4.39, arrow 1). When the callback to **UnSecApp.Exe** is performed (see Figure 4.39, arrow 3), the wrapper intercepts the callback as well to extract the SID of the caller. The extraction of the SID implies a call against Active Directory, which validates the SID existence of the caller. Next, the SID of the caller is compared with the SID of the client (see Figure 4.39, arrow 4) and if they match, **UnSecApp.Exe** executes the callback to the client application (see Figure 4.39, arrow 6).

As long as the script runs in an authenticated security context, the activation doesn't affect the operation. However, if the callback is delivered under a different identity, the callback will not be returned to the sink routine, and **UnSecApp.Exe** will place an entry into the event log. However, the activation of the registry key setting could only affect applications in pre-Windows 2000 environments due to the lack of LocalSystem identity.

Note: LocalSystem in a Kerberos environment has an identity in an Active Directory environment like any other users. For instance, this allows the Windows services to run in the machine security context rather than a specific domain user. In NTLM environments, LocalSystem has local credentials only and cannot be resolved in a domain, which therefore doesn't provide a network identity.

4.14.2 Setting the security descriptor of a CIM repository namespace

When creating a namespace in the CIM repository, the only element of information required is the name of the namespace, since the name of the namespace is used as a key property of the instance representing the namespace. However, every namespace is protected by a security descriptor, which is inherited from the parent namespace at creation time. Since the security of a namespace could be different from the one set on the parent, it is possible to set a different security descriptor (i.e., by using the *SetSD* method exposed by the *__SystemSecurity* system class). Since the security initiative, it is now impossible to set a security descriptor to null.

If by any chance the security descriptor is set to null on a namespace, the namespace will inherit the security of the parent namespace, since the "non-inheritance" option is not set. However, by eliminating the ability to set a null security descriptor, Microsoft wants to prevent anyone from deliberately removing the namespace security descriptor. Therefore, this initiative is purely preventive.

4.14.3 The ADSI WMI Extension

In Chapter 5 of the first book, *Understanding WMI Scripting*, we see how the ADSI WMI Extension can be used from scripts. This extension is available under Windows NT 4.0 (if ADSI and WMI are installed), Windows 2000, and Windows XP. Under Windows Server 2003 with the security

push initiative, this extension has been removed from the Operating System. At writing time, the availability of this extension as a separate download is still under discussion at Microsoft. Actually, Microsoft decided to remove this extension from the Operating System for two reasons:

- One of the drivers during the security push was to leave only enabled components that are intensively used. Since the ADSI WMI Extension was not heavily used, Microsoft decided to remove it from the Operating System to reduce the potential surface of attacks.
- Although the ADSI WMI Extension is not subject to any particular security thread, Microsoft decided to remove it, since it requires high-level privileges to run.

4.14.4 The Windows Authorizations Access Group (WAAG)

The Windows Authorizations Access Group (WAAG) is a new built-in security group introduced in Windows Server 2003. The members of this group are allowed to look up group membership for a particular user. For example, the idea is that user "Alain Lissoir" should not be able to look up other users in the Active Directory and find out that they are members of some selected group of people created by someone else. Therefore, under Windows Server 2003, looking up group membership for another user has to be explicitly granted to users by adding them to the WAAG group in the domain. How is this group related to WMI? Actually, when someone creates a subscription (i.e., for a permanent event consumer), WMI needs to verify if the user who creates the subscriptions has the rights to do that. This verification process enumerates all group membership for the users and therefore the account under which this enumeration is done must be included in WAAG built-in group.

4.15 Summary

In this chapter, besides the IIS configuration regarding WMI and the WMI security changes driven by the security push initiative, we discussed how security descriptors are structured and how they can be deciphered, based on:

- The selected Operating System (Windows NT 4.0/Windows 2000 versus Windows XP/Windows Server 2003)
- The security descriptor origin (file, folders, Active Directory, etc.)

- The security descriptor object model representation (WMI versus ADSI)

The management of the security descriptor clearly shows the limitation of WMI in respect to some secured objects. The complementary nature of ADSI and CDOEXM are, of course, a great help in offering good management coverage of most common security descriptors. However, this requires an extended knowledge of these two technologies, which complicates the scripting techniques. Hopefully, to minimize the learning curve of WMI, ADSI, and CDOEXM, the **WMIManageSD.Wsf** script is generic enough to easily extend its security descriptor management capabilities to support other securable objects, on the condition that some COM components are available to offer a scriptable access.