

The WMI Providers

3.1 Objective

Although the *Win32* provider class capabilities cover a lot of material, we haven't examined WMI providers that support various components and applications, which are part of Windows Server 2003 and the previous Windows platforms. Under Windows Server 2003, WMI comes with more than 40 other providers supporting a specific set of classes. We must make a clear distinction between the providers that are part of the system installation and the ones that can be added during an optional software component installation. In this chapter, we will only focus on the WMI providers provided with the Windows Server 2003 installation. For example, this includes providers such as the *IP routing* providers, the *DNS* providers, the *Active Directory* providers, and the *Resultant Set of Policies* (RSOP) providers, to name a few. WMI providers related to optional Windows Server 2003 components or applications, such as *Internet Information Server* (IIS) provider, *Network Load Balancing* (NLB) provider, and the *Cluster Service* provider, will be examined later in the book, as will certain WMI-enabled applications such as Exchange 2000, HP OpenView Operations for Windows, Microsoft Operation Manager, and the Windows Server 2003 Framework.

3.2 The WMI providers

Each of the providers examined in this chapter can be considered as specialized providers. Some of them provide functionality about system components not supported by the *Win32* providers, such as the *Windows Product Activation* provider and the *Active Directory Trust Monitoring* provider. On the other hand, some providers complement functionality supported by the *Win32* providers. For example, in the previous chapter, when we examined the classes related to the file system, we saw that the *CIM_DataFile* class is

supported by the *Win32* provider. Even if the *CIM_DataFile* class is a superclass of the *Win32_NTEventLogFile* class, the *Win32_NTEventLogFile* class is supported by a set of specialized providers, which are designed to manage the Windows NT Event Log files: the *NT Event Log* providers.

All WMI providers are designed to manage a particular Windows Server 2003 component and therefore implement some capabilities relevant to that component. For example, the *Ping* provider is implemented as an instance provider and retrieves the ping command status code when pinging a host. The *DNS* provider is implemented as an instance provider to manipulate DNS zones and DNS records stored in these zones. Another example is the *Power management* provider, which is able to trigger an event when the power state of the system changes.

To ease the discovery of the WMI providers available, they are classified in different categories, which will be examined section by section in this chapter. These categories are:

- **Core OS components providers:** This category contains WMI providers that manage Operating System components such as the NT Event Logs, the registry, and the trusts.
- **Core OS components event providers:** This category contains WMI providers that trigger events when something occurs in the Operating System. The *Power management* and the *Shutdown* WMI event providers are just two examples of providers in this category.
- **Core OS file system components providers:** This category contains providers managing specific features of the file system, such as disk quotas or the Distributed File System (DFS).
- **Active Directory components providers:** The WMI providers in this category only support features related to Active Directory. For example, accessing objects in Active Directory or monitoring Active Directory Replication are two features offered by the WMI providers of this category.
- **Network components providers:** This category only contains the providers supporting features related to network components. For example, in this category we find WMI providers that manage DNS, the IP routing table, and SNMP devices.
- **Performance providers:** This category contains providers giving access to performance counter information. Performance counters exist for applications as well as the Operating System, but we will only focus on the latter in this category.

- **Helper providers:** The providers included in this category are the specialized WMI providers that are not directly managing real-world entities but are helping to access some information in the CIM repository from a local or a remote system. The *View* provider and the *Forwarding* consumer provider are in this category.

Note that some of the providers examined in this chapter are also available under Windows 2000 and Windows NT! During the WMI provider discussion, we will mention in which operating system they are available.

Let's start the WMI providers discovery!

3.3 Core OS components providers

3.3.1 The WBEM provider

The *WBEM* provider exposes the WMI configuration parameters. The three classes supported by this provider are listed in Table 3.1.

Table 3.1

The WBEM Provider Classes

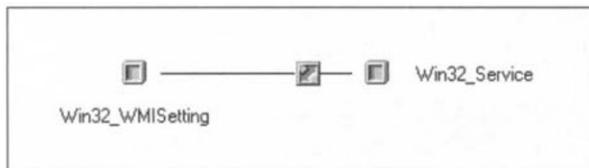
Name	Type	Comments
Win32_WMISetting	Dynamic (Singleton)	Contains the operational parameters for the WMI service.
Win32_WMIElementSetting	Association	Association between a service running in the Win32 system, and the WMI settings it can use.
Win32_MethodParameterClass	Abstract	Abstract, base class that implements method parameters derived from this class.

You can use the *LoadCIMInXL.wsf* script (see Sample 4.32 in the appendix) or the **WMI CIM Studio** of the Platform SDK to gather more information about the class properties. In the Platform SDK, these classes are classified as the WMI System Management classes.

The *WBEM* provider classes are available in the **Root\CMIV2** namespace. Basically, the *Win32_WMISetting* class gives access to the WMI settings available from the WMI Control MMC (in the Computer Management MMC). Since there is one installation of WMI per system, there is only one instance of the *Win32_WMISetting* class. This is the reason why this class has no Key property defined in its properties and is therefore defined as a singleton class.

Because this class relates to one WMI installation in one system, the *Win32_WMIElementSetting* class associates the *Win32_WMISetting* with the *Win32_Service* class (see Figure 3.1). The *Win32_WMISetting* has one Windows associated service instance: the **WinMgmt.Exe** service.

Figure 3.1
The Win32_WMISetting class associations.



The *Win32_MethodParameterClass* class is an abstract class, which means that no instance can be retrieved from the class. This class is used as a template for classes used as parameters when invoking specific methods. The *Win32_MethodParameterClass* is a superclass for the *Win32_SecurityDescriptor*, *Win32_ACE*, *Win32_Trustee*, and *Win32_ProcessStartup* classes. For example, when executing the *Create* method of the *Win32_Process* class (see Sample 2.53 in Chapter 2), we create a *Win32_ProcessStartup* instance to define some process parameters used for its creation.

The *WBEM* provider is an instance provider only. If the *Win32_WMISetting* class must be used in the context of a WQL event query, the **WITHIN** statement must be used, because the provider is not implemented as an event provider. For example, a valid WQL query for the *WBEM* provider would be:

```
Select * From __InstanceModificationEvent Within 10 Where TargetInstance ISA 'Win32_WMISetting'
```

The *WBEM* instance provider supports the Get, Put, and Enumeration operations (see Table 3.2). This means that it is possible to retrieve a single instance (get) with its WMI object path, modify a single instance (put), and retrieve a collection of instances (enumeration).

Table 3.2 The WBEM 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
WBEM core Provider	Root/CIMV2	X						X X X				X X X X X				

The next script sample is able to change some of the WMI settings from the command line. The WMI settings that can be changed are available from the script command-line help.

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

Usage: WMISettings.wsf [/Action:value [/ASPScriptDefaultNamespace:value] [/ASPScriptEnabled[+|-]]
                      [/BackupInterval:value] [/EnableEvents[+|-]] [/LoggingDirectory:value]
                      [/LoggingLevel:value] [/MaxLogFileSize:value]
                      [/Machine:value] [/User:value] [/Password:value]

Options:

Action           : Specify the operation to perform: [List] or [Config].
ASPScriptDefaultNamespace : Defines the namespace used by calls from the scripting API if none is
                           specified by the caller.
ASPScriptEnabled   : Indicates whether WMI scripting can be used on Active Server Pages
                     (ASP). This property is valid on Windows NT 4.0 systems only.
BackupInterval     : Specifies the length of time that will elapse between backups of the WMI
                     database.
EnableEvents       : Indicates whether the WMI event subsystem should be enabled.
LoggingDirectory   : Specifies the directory path containing the location of the WMI system
                     log files.
LoggingLevel        : Indicates whether event logging is enabled and the verbosity level of
                     logging used.
MaxLogFileSize     : Indicates the maximum size of the log files produced by the WMI service.
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)

Examples:
```

```
WMISettings.wsf /Action>List
WMISettings.wsf /Action:Config /ASPScriptDefaultNamespace:Root\CMV2
WMISettings.wsf /Action:Config /ASPScriptEnabled+
WMISettings.wsf /Action:Config /BackupInterval:30
WMISettings.wsf /Action:Config /EnableEvents+
WMISettings.wsf /Action:Config /LoggingDirectory:C:\WINDOWS\System32\Wbem\Logs
WMISettings.wsf /Action:Config /LoggingLevel:ErrorsOnly
WMISettings.wsf /Action:Config /MaxLogFileSize:65535
```

The *Win32_WMISetting* class exposes more properties than the ones that can be modified by the script. However, this sample can be easily extended to cater to any additional WMI settings that need to be modified. The purpose of the script is to view the *Win32_WMISetting* singleton instance and update some of its properties. From a scripting technique point of view, there is nothing new compared with all scripts previously developed. The segment of code changing some WMI settings is shown in Sample 3.1.

→ **Sample 3.1** *Updating the WMI settings*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:    <runtime>
```

```
...  
38:    </runtime>  
39:  
40:    <script language="VBScript" src=..\Functions\DisplayFormattedPropertyFunction.vbs" />  
41:    <script language="VBScript" src=..\Functions\TinyErrorHandler.vbs" />  
42:  
43:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>  
44:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />  
45:  
46:    <script language="VBScript">  
47:        <![CDATA[  
...  
51:        Const cComputerName = "LocalHost"  
52:        Const cWMINamespace = "Root/cimv2"  
53:        Const cWMIClass = "Win32_WMISetting"  
...  
77:        ' -----  
78:        ' Parse the command line parameters  
79:        If WScript.Arguments.Named.Count = 0 Then  
80:            WScript.Arguments.ShowUsage()  
81:            WScript.Quit  
82:        End If  
...  
131:        If Len(strComputerName) = 0 Then strComputerName = cComputerName  
132:  
133:        objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault  
134:        objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate  
135:  
136:        Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINamespace, _  
137:                                         strUserID, strPassword)  
...  
140:        Set objWMIInstance = objWMIServices.Get (cWMIClass & "=@")  
...  
143:        ' -- LIST -----  
144:        If boolList = True Then  
145:            Set objWMIPropertySet = objWMIInstance.Properties_  
146:            For Each objWMIProperty In objWMIPropertySet  
147:                DisplayFormattedProperty objWMIInstance, _  
148:                                         " " & objWMIProperty.Name, _  
149:                                         objWMIProperty.Name, _  
150:                                         Null  
151:            Next  
152:            Set objWMIPropertySet = Nothing  
153:            WScript.Echo  
154:        End If  
155:  
156:        ' -- CONFIG -----  
157:        If boolConfig = True Then  
158:  
159:            If Len (strASPScriptDefaultNamespace) Then  
160:                objWMIInstance.ASPScriptDefaultNamespace = strASPScriptDefaultNamespace  
161:            End If  
162:  
163:            If Len (boolASPScriptEnabled) Then  
164:                objWMIInstance.ASPScriptEnabled = boolASPScriptEnabled  
165:            End If  
166:  
167:            If intBackupInterval <> 0 Then  
168:                objWMIInstance.BackupInterval = intBackupInterval  
169:            End If  
170:
```

```
171:     If Len (boolEnableEvents) Then
172:         objWMIInstance.EnableEvents = boolEnableEvents
173:     End If
174:
175:     If Len (strLoggingDirectory) Then
176:         objWMIInstance.LoggingDirectory = strLoggingDirectory
177:     End If
178:
179:     If longMaxLogFileSize <> 0 Then
180:         objWMIInstance.MaxLogFileSize = longMaxLogFileSize
181:     End If
182:
183:     If Len (strLoggingLevel) Then
184:         objWMIInstance.LoggingLevel = intLoggingLevel
185:     End If
186:
187:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _
188:                         wbemFlagReturnWhenComplete)
...
191:     WScript.Echo "WMI Settings updated."
192:
193: End If
...
199: ]]>
200: </script>
201: </job>
202:</package>
```

As previously mentioned, the *Win32_WMISetting* class is a singleton class; this means that:

- Only one instance is available per system.
- There is no Key property.

Note that a particular syntax is used to create an instance of the *Win32_WMISetting* singleton class (line 140). The WMI settings are updated from line 157 through 193. Only the specified settings are modified. Once the desired WMI properties are updated, the script commits the changes back to the system (lines 187 and 188).

The Sample 3.1 output obtained with the **/Action>List** switch is as follows:

```
1: C:\>WMISettings.wsf /Action>List
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: ASPScriptDefaultNamespace: ..... Root\cimv2
6: ASPScriptEnabled: ..... TRUE
7: AutorecoverMofs: ..... J:\WINDOWS\system32\WBEM\cimwin32.mof
8:                               J:\WINDOWS\system32\WBEM\cimwin32.mfl
9:                               J:\WINDOWS\system32\WBEM\system.mof
J:\WINDOWS\system32\WBEM\wmipcima.mof
J:\WINDOWS\system32\WBEM\wmipcima.mfl
10:
11:
...
...
...
```

```

62:                                         J:\WINDOWS\system32\WBEM\snmpsmir.mof
63:                                         J:\WINDOWS\system32\WBEM\snmpreg.mof
64:                                         J:\WINDOWS\System32\Wbem\RSOp.mof
65:                                         J:\WINDOWS\System32\Wbem\RSOp.mfl
66:                                         J:\WINDOWS\System32\Wbem\SceRsop.mof
67:                                         J:\WINDOWS\system32\wbem\iiswmi.mfl
68:                                         J:\WINDOWS\system32\wbem\ADStatus\TrustMon.mof
69:                                         J:\WINDOWS\System32\replprov.mof
70: BackupInterval: ..... 30
71: BuildVersion: ..... 3663.0000
73: EnableEvents: ..... TRUE
74: EnableStartupHeapPreallocation: ..... FALSE
75: HighThresholdOnClientObjects: ..... 20000000
76: HighThresholdOnEvents: ..... 20000000
77: InstallationDirectory: ..... J:\WINDOWS\system32\WBEM
78: LoggingDirectory: ..... J:\WINDOWS\system32\WBEM\Logs\
79: LoggingLevel: ..... 1
80: LowThresholdOnClientObjects: ..... 10000000
81: LowThresholdOnEvents: ..... 10000000
82: MaxLogFileSize: ..... 65536
83: MaxWaitOnClientObjects: ..... 60000
84: MaxWaitOnEvents: ..... 2000
85: MofSelfInstallDirectory: ..... J:\WINDOWS\system32\WBEM\MOF

```

Among this list of properties, the output shows the list of MOF files to recompile when recovering the CIM repository (lines 7 through 69), the backup interval (line 70), and the WMI logging level (line 79).

3.3.2 NT Event Log providers

The *NT Event Log* providers consist of two WMI providers: one WMI event provider and one instance and method provider, which support the *get*, *put*, and *enumeration* operations (see Table 3.3).

Table 3.3 The NT Event Log Providers 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
Event Log Providers																
MS_NT_EVENTLOG_EVENT_PROVIDER	Root/CIMV2					X						X	X	X	X	X
MS_NT_EVENTLOG_PROVIDER	Root/CIMV2	X	X				X	X	X			X	X	X	X	X

These providers are designed to expose information stored in the Windows NT Event Log files, view the configuration settings related to the Windows NT Event Log files, and perform some specific actions, such as

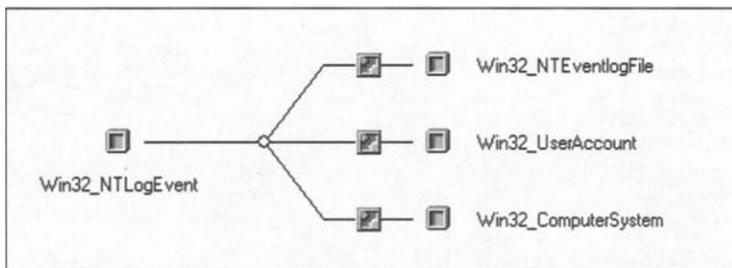
clearing or backing up the NT Event Log. The classes supported by these providers are listed in Table 3.4.

→ **Table 3.4** *The NT Event Log Providers Classes*

Name	Type	Comments
Win32_NTEventLogFile	Dynamic	Represents settings of a Windows NT/Windows 2000 Event Log file.
Win32_NTLogEvent	Dynamic	Represents data stored in a Windows NT/Windows 2000 log file.
Win32_NTLogEventComputer	Association	Relates instances of Win32_NTLogEvent and Win32_ComputerSystem.
Win32_NTLogEventLog	Association	Relates instances of Win32_NTLogEvent and Win32_NTEventLogFile
Win32_NTLogEventUser	Association	Relates instances of Win32_NTLogEvent and Win32_UserAccount.

These classes are available in the Root\CMV2 namespace. The *Win32_NTLogEvent* class is shown in Figure 3.2.

→ **Figure 3.2**
The *Win32_NTLogEvent* associations.



Since there is an *NT Event Log* event provider, there is no need to use the **WITHIN** statement in a WQL event query. In such a case, a valid WQL query would be as follows:

- To capture any WMI event corresponding to a record addition to any NT Event Log:

```
Select * From __InstanceCreationEvent Where TargetInstance ISA 'Win32_NTLogEvent'
```

- To capture any WMI event corresponding to a record addition to the Directory Service Event Log:

```
Select * From __InstanceCreationEvent Where TargetInstance ISA 'Win32_NTLogEvent'
And TargetInstance.LogFile='Directory Service'
```

It is important to note that the *NT Event Log* event provider only supports the intrinsic event *__InstanceCreationEvent* class. The generic asynchronous script to capture events from a WQL event query passed on the command line (see Sample 6.17, “A generic script for asynchronous event notification” in the appendix) can be used with the following command line. The output will be as follows when a new Event Log record is created:

```

1:  C:\>GenericEventAsyncConsumer.wsf "Select * From __InstanceCreationEvent
2:                                Where TargetInstance ISA 'Win32_NTLogEvent' And
3:                                TargetInstance.EventCode=1013"
4: Microsoft (R) Windows Script Host Version 5.6
5: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
6:
7: Waiting for events...
8:
9: BEGIN - OnObjectReady.
10: Sunday, 11 November, 2001 at 16:51:01: '__InstanceCreationEvent' has been triggered.
11:
12: - __InstanceCreationEvent -----
13: SECURITY_DESCRIPTOR: ..... 1,0,4,128,84,1,0,0,96,1,0,0,0,0,0,0,20,...
14:
15: - Win32_NTLogEvent -----
16: Category: ..... 1
17: CategoryString: ..... Knowledge Consistency Checker
18: ComputerName: ..... NET-DOPEN6400A
19: EventCode: ..... 1013
20: EventIdentifier: ..... 1073742837
21: EventType: ..... 3
22: *LogFile: ..... Directory Service
23: Message: ..... Internal event: The replication topology ...
24: *RecordNumber: ..... 72
25: SourceName: ..... NTDS KCC
26: TimeGenerated: ..... 11-11-2001 16:50:39
27: TimeWritten: ..... 11-11-2001 16:50:39
28: Type: ..... information
29: User: ..... NT AUTHORITY\ANONYMOUS LOGON
30:
31: TIME_CREATED: ..... 11-11-2001 15:50:40 (2001111145040.276894+060)
32:
33: END - OnObjectReady.
34:
35: Cancelling event subscription ...
36:
37: BEGIN - OnCompleted.
38: END - OnCompleted.
39: Finished.

```

The second interesting class supported by the *NT Event Log* providers is the *Win32_NTEventLogFile* class. In addition to being able to review and configure various NT Event Log configuration settings (i.e., Maximum Size, overwrite policy) you can also clear and back up any NT Event Log. The next script sample illustrates how to achieve this. Its command-line parameters are as follows:

```

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

Usage: WMIEventLog.wsf [/Action:value] [/EventLog:value] [/MaxFileSize:value]
      [/OverwriteOutDated:value] [/Machine:value] [/User:value] [/Password:value]

Options:

Action      : Specify the operation to perform: [List], [Config], [Clear] and [Backup].

```

```

EventLog      : Specify the Log name (All means all available Event Logs).
MaxFileSize   : Set the maximum size of the Event Log file (in Kb).
OverwriteOutDated : Overwrites events policy [WhenNeeded], [Never] or a value 1 and 365.
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)
Examples:

```

```

WMIEventLog.wsf /Action>List
WMIEventLog.wsf /Action>Clear /EventLog>All
WMIEventLog.wsf /Action>Clear /EventLog:NTDS
WMIEventLog.wsf /Action>Backup /EventLog:DNSEvent
WMIEventLog.wsf /Action>Backup /EventLog:NTFRS
WMIEventLog.wsf /Action>Config /EventLog:SecEvent /MaxFileSize:5120
WMIEventLog.wsf /Action>Config /EventLog:SysEvent /OverwriteOutDated:7
WMIEventLog.wsf /Action>Config /EventLog>All /OverwriteOutDated:WhenNeeded
WMIEventLog.wsf /Action>Config /EventLog:AppEvent /OverwriteOutDated:Never
WMIEventLog.wsf /Action>Config /EventLog>All /MaxFileSize:5120 /OverwriteOutDated:WhenNeeded

```

The script code using the *Win32_NTEventLogFile* class capabilities is shown in Sample 3.2. As usual, the script uses the same structure to define (skipped lines 13 through 35) and parse the command-line parameters (skipped lines 78 through 142) and to list the instance properties available from this class (lines 159 through 173).

Sample 3.2

Viewing and updating the NT Event Log configuration and clearing and backing up the NT Event Log information

```

1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
35:  </runtime>
36:
37:  <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
38:  <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
39:
40:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
41:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
42:
43:  <script language="VBScript">
44:  <![CDATA[
.:
48:  Const cComputerName = "localhost"
49:  Const cWMINNameSpace = "Root/cimv2"
50:  Const cWMIClass = "Win32_NTEventLogFile"
.:
77:  -----
78:  ' Parse the command line parameters
79:  If WScript.Arguments.Named.Count = 0 Then
80:    WScript.Arguments.ShowUsage()
81:    WScript.Quit
82:  Else

```

```
83:      Select Case Ucase(WScript.Arguments.Named("Action"))
...:
96:      End Select
97:  End If
...:
116:  If Len(strOverWriteOutDated) Then
117:      Select Case Ucase (strOverWriteOutDated)
118:          Case "WHENNEEDED"
119:              intOverWriteOutDated = 0
120:              strOverWriteOutDated = "WhenNeeded"
121:          Case "NEVER"
122:              intOverWriteOutDated = 4294967295      ' 2^32 - 1
123:              strOverWriteOutDated = "Never"
124:          Case Else
125:              intOverWriteOutDated = Cint (strOverWriteOutDated)
126:              If intOverWriteOutDated < 1 Or intOverWriteOutDated > 365 Then
127:                  WScript.Echo "Invalid overwrite outdated parameter." & vbCrLf
128:                  WScript.Arguments.ShowUsage()
129:                  WScript.Quit
130:              End If
131:              strOverWriteOutDated = "OutDated"
132:      End Select
133:  End If
...:
142:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
143:
144:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
145:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
146:  objWMILocator.Security_.Privileges.AddAsString "SeBackupPrivilege", True
147:  objWMILocator.Security_.Privileges.AddAsString "SeSecurityPrivilege", True
148:
149:  Set objWMIProperties = objWMILocator.ConnectServer(strComputerName, cWMINamespace, _
150:                                         strUserID, strPassword)
...:
153:  Set objWMIInstances = objWMIProperties.InstancesOf (cWMIClass, wbemFlagUseAmendedQualifiers)
...:
156:  For Each objWMIInstance In objWMIInstances
157:
158:      ' -- LIST -----
159:      If boolList = True Then
160:          WScript.Echo "-- " & objWMIInstance.LogfileName & " Event Log " & _
161:              "(" & Ucase (objWMIInstance.FileName) & _
162:              ")" & String (60, "-")
163:
164:          Set objWMIPropertySet = objWMIInstance.Properties_
165:          For Each objWMIProperty In objWMIPropertySet
166:              DisplayFormattedProperty objWMIInstance, _
167:                  " " & objWMIProperty.Name, _
168:                  objWMIProperty.Name, _
169:                  Null
170:          Next
171:          Set objWMIPropertySet = Nothing
172:          WScript.Echo
173:      End If
174:
175:      If UCASE (objWMIInstance.FileName) = strEventLogName Or _
176:          strEventLogName = "ALL" Then
177:          ' -- CONFIG -----
178:          If boolConfig = True Then
179:
```

```
180:         If intMaxFileSize <> 0 Then
181:             objWMIInstance.MaxFileSize = intMaxFileSize * 1024
182:         End If
183:
184:         If Len (strOverWriteOutDated) Then
185:             objWMIInstance.OverwriteOutDated = intOverWriteOutDated
186:             objWMIInstance.OverWritePolicy = strOverWriteOutDated
187:         End If
188:
189:         objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _
190:                             wbemFlagReturnWhenComplete Or _
191:                             wbemFlagUseAmendedQualifiers)
192:     If Err.Number Then ErrorHandler (Err)
193:
194:     If intMaxFileSize <> 0 Then
195:         varTemp = objWMIInstance.LogfileName & _
196:                     " Event Log maximum size is set on " & _
197:                     intMaxFileSize & " Kb. "
198:     Else
199:         varTemp = objWMIInstance.LogfileName & _
200:                     " Event Log overwrite policy configured. "
201:     End If
202:
203:     Select Case Ucase (strOverWriteOutDated)
204:         Case "WHENNEEDED"
205:             WScript.Echo varTemp & "Events are overwritten as needed."
206:         Case "NEVER"
207:             WScript.Echo varTemp & "Events are never overwritten."
208:         Case "OUTDATED"
209:             WScript.Echo varTemp & "Events older than " & _
210:                             intOverWriteOutDated & _
211:                             " day(s) are overwritten."
212:         Case Else
213:             WScript.Echo varTemp
214:     End Select
...
217:     End If
218:
219:     ' -- CLEAR -----
220:     If boolClear = True Then
221:         intRC = objWMIInstance.ClearEventlog
222:     If Err.Number Then ErrorHandler (Err)
223:
224:         WScript.Echo objWMIInstance.LogfileName & _
225:                     " Event Log is cleared (" & intRC & ")."
...
228:     End If
229:
230:     ' -- BACKUP -----
231:     If boolBackup = True Then
232:         intRC = objWMIInstance.BackupEventlog (objWMIInstance.Name & ".Bak")
233:     If Err.Number Then ErrorHandler (Err)
234:
235:         WScript.Echo objWMIInstance.LogfileName & " Event Log is backuped to '" & _
236:                         Ucase (objWMIInstance.Name) & ".Bak' (" & intRC & ")."
...
239:     End If
240:
241:     End If
242: Next
```

```
...:  
248:    ]]>  
249:    </script>  
250:  </job>  
251:</package>
```

The most interesting part of this script resides in the NT Event Log settings configuration (lines 178 through 217) and the method invocations (lines 220 through 228 for clearing and lines 231 through 239 for backup). The script is structured in such a way that it can manage all NT Event Logs in one single run. For this, the script gets the collection of all NT Event Logs available (line 153) and performs an enumeration (lines 156 through 242). Then, the script uses a special NT Event Log name called "All" (as defined in the script at line 176) to determine if the operation concerns a specific NT Event Log name or all the collection (lines 175 and 176). This script uses the "All" label to differentiate the scope of the requested operation.

An interesting NT Event Log setting is the overwrite policy. To set the overwrite policy (lines 185 and 186) two properties of the NT Event Log instance must be set up. The two property values are determined during the command-line parameters parsing (lines 117 through 132). The command-line parsing for this setting ensures that correct values are set. It is important to note that both property values must be set up as shown in Table 3.5.

Table 3.5

The OverWritePolicy and OverwriteOutDated Property Values

	WhenNeeded	Never	OutDated
OverWritePolicy	0	4294967295	1 < Value > 365
OverwriteOutDated	WhenNeeded	Never	OutDated

Last but not least, some privileges at the WMI level must be specified (lines 146 and 147) to ensure transparent access to the Security NT Event Log (line 146) and perform the backup operation (line 147).

3.3.3 Registry providers

There are three WMI providers in this category: one instance and method provider (*RegProv*), one property provider (*RegPropProv*), and one event provider (*RegistryEventProvider*). The providers capabilities are summarized in Table 3.6.

→ **Table 3.6** *The Registry Providers 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
Registry Providers																
RegistryEventProvider	Root\DEFAULT		X									X X X X X				
RegPropProv	Root\DEFAULT	X X					X X X					X X X X X				

These providers are designed to expose information stored in the Windows NT Registry. The classes supported by these providers are listed in Table 3.7.

→ **Table 3.7** *The Registry Providers Classes*

Name	Type	Comments
StdRegProv	Dynamic	The StdRegProv class only contains methods that interact with the system registry. You can use these methods to verify the access permissions for a user, create, enumerate, and delete registry keys; create, enumerate, and delete named values; and read, write, and delete data values.
RegistryValueChangeEvent	Extrinsic event	Represents a registry extrinsic event that corresponds to a registry value modification.
RegistryKeyChangeEvent	Extrinsic event	Represents a registry extrinsic event that corresponds to a registry key modification.
RegistryTreeChangeEvent	Extrinsic event	Represents a registry extrinsic event that corresponds to a registry tree modification.

These classes are available in the Root\Default namespace. The *StdRegProv* class only exposes methods; it does not expose any property and has no association.

The *Registry* event provider allows a process to receive a notification when a registry change occurs. For this, three extrinsic event classes are supported (see Table 3.7). Each of these classes corresponds to a particular modification made in the registry. For example, imagine that you would like to monitor the registry key value that enables Active Directory schema changes. The key value is named “Schema Update Allowed” and is located in the following registry key hive:

HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters

To detect any change made on that key value, the following WQL event query must be used:

```
Select * FROM RegistryValueChangeEvent Where Hive='HKEY_LOCAL_MACHINE' AND
KeyPath='SYSTEM\CurrentControlSet\\Services\NTDS\Parameters' AND
ValueName='Schema Update Allowed'
```

Now, if you want to detect all changes made to any key values below the same registry key hive, the following WQL Event query must be used:

```
Select * FROM RegistryKeyChangeEvent Where Hive='HKEY_LOCAL_MACHINE' AND
KeyPath='SYSTEM\CurrentControlSet\Services\NTDS\Parameters'
```

If you want to detect all changes made to any key values under the following registry key hive and all child registry entries, use the following:

HKLM\SYSTEM\CurrentControlSet\Services\NTDS

The following WQL event query must be used:

```
Select * FROM RegistryTreeChangeEvent Where Hive='HKEY_LOCAL_MACHINE' AND
RootPath='SYSTEM\CurrentControlSet\Services\NTDS'
```

To summarize:

- The extrinsic *RegistryValueChangeEvent* class focuses the change detection on a particular key value.
- The extrinsic *RegistryKeyChangeEvent* class focuses the change detection on all key values below a given registry key hive.
- The extrinsic *RegistryTreeChangeEvent* class focuses the change detection on all key values in a given registry key hive, as shown in the example below.

If we reuse the generic asynchronous script to capture events from a WQL event query (see Sample 6.17, “A generic script for asynchronous event notification” in the appendix) with the last WQL Event query sample, the output will be as follows:

```
1: C:\>GenericEventAsyncConsumer.wsf "Select * FROM RegistryTreeChangeEvent Where
2:                                         Hive='HKEY_LOCAL_MACHINE' AND
3:                                         RootPath='SYSTEM\CurrentControlSet\services\NTDS'"
4:                                         /NameSpace:Root\Default
5: Microsoft (R) Windows Script Host Version 5.6
6: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
7:
8: Waiting for events...
9:
10: BEGIN - OnObjectReady.
11: Monday, 19 November, 2001 at 13:27:29: 'RegistryTreeChangeEvent' has been triggered.
12:
13: - RegistryTreeChangeEvent -----
14:   Hive: ..... HKEY_LOCAL_MACHINE
15:   RootPath: ..... SYSTEM\CurrentControlSet\services\NTDS
16:   TIME_CREATED: ..... 19-11-2001 12:27:29 (20011119112729.796836+060)
17:
18: END - OnObjectReady.
```

Even if we get a notification, it is important to note that we do not get the registry value associated with the change in the instance representing the notification.

Besides the registry key monitoring, the methods exposed by the *StdRegProv* class allow the enumeration of registry keys and values. It is also possible to create new registry keys, read and update registry key values, and delete registry keys and values. These methods are summarized in Table 3.8.

Table 3.8*The StdRegProv Methods*

Method name	Description
CheckAccess	Verifies that the user possesses the specified access permissions.
CreateKey	Creates a subkey.
DeleteKey	Deletes a subkey.
DeleteValue	Deletes a named value.
EnumKey	Enumerates subkeys.
EnumValues	Enumerates the named values of a key.
GetBinaryValue	Gets the binary data value of a named value.
GetDWORDValue	Gets the DWORD data value of a named value.
GetExpandedStringValue	Gets the expanded string data value of a named value.
GetMultiStringValue	Gets the multiple string data values of a named value.
GetStringValue	Gets the string data value of a named value.
SetBinaryValue	Sets the binary data value of a named value.
SetDWORDValue	Sets the DWORD data value of a named value.
SetExpandedStringValue	Sets the expanded string data value of a named value.
SetMultiStringValue	Sets the multiple string values of a named value.
SetStringValue	Sets the string value of a named value.

Sample 3.3 illustrates the use of these methods. Its command-line parameters are as follows:

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

Usage: WMIRegistry.wsf [RegistryKeyValue] /BaseKey:value /Action:value [/DisplayRegValues[+|-]]
          [/KeyType:value] [/SearchString:value]
          [/ReplaceString:value] [/Machine:value]
          [/User:value] [/Password:value]

Options:

RegistryKeyValue : The registry key name with its value.
BaseKey         : Specify the base registry.
Action          : Specify the operation to perform: [list], [Search], [Replace], [create], [update] or
[delete].
DisplayRegValues : Display the registry key values (default=TRUE)
KeyType         : Specify the registry value type.
SearchString    : String to search in the registry.
ReplaceString   : String to replace in the registry.
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)

Examples:
```

```
WMIRegistry.wsf /BaseKey:HKLM\Software\MyKeys /Action:Create
WMIRegistry.wsf MyStringValue=XYW /KeyType:REG_SZ /BaseKey:HKLM\Software\MyKeys /Action:Create
WMIRegistry.wsf MyMultiStringValue=RST,STU,TUV,UWW,VWX,WXY,XYZ /KeyType:REG_MULTI_SZ
/BaseKey:HKLM\Software\MyKeys /Action:Create
WMIRegistry.wsf MyExpandedStringValue=%SystemRoot%\System32\Wbem /KeyType:REG_EXPAND_SZ
/BaseKey:HKLM\Software\MyKeys /Action:Create
WMIRegistry.wsf MyDWordValue=436327632 /KeyType:REG_DWORD
/BaseKey:HKLM\Software\MyKeys /Action:Create
WMIRegistry.wsf MyBinaryValue=03,04,255,78,34,23 /KeyType:REG_BINARY
/BaseKey:HKLM\Software\MyKeys /Action:Create

WMIRegistry.wsf /BaseKey:HKLM\Software /Action>List /DisplayRegValues-
WMIRegistry.wsf /BaseKey:HKLM\Software /Action>List /DisplayRegValues+
WMIRegistry.wsf /BaseKey:HKLM\SOFTWARE\MyKeys /Action:Search /SearchString:String
WMIRegistry.wsf /BaseKey:HKLM\SOFTWARE\MyKeys /Action:Replace
/SearchString:String /ReplaceString:MyString

WMIRegistry.wsf MyBinaryValue /BaseKey:HKLM\Software\MyKeys /Action:Delete
WMIRegistry.wsf /BaseKey:HKLM\Software\MyKeys /Action:Delete
```

With this script it is possible to manage the Windows registry from the command line. Other than simply browsing or updating the registry, it is also capable of performing search and replace operations in the registry. This script is shown in Samples 3.3 through 3.9. Careful use of the replace feature is recommended, since it is very easy to mess up a working system!

To perform the command-line definition (skipped lines 13 through 42) and parsing (lines 111 through 244), the script reuses the same structure as seen in the previous script samples. Next, it reuses most of the functions previously developed (lines 46 through 48). Only two new functions, SearchString() (line 44) and ReplaceString() (line 45), are added to support the search and replace capabilities. We will come back to the use of these functions later, along with the script discovery (see Samples 3.3 through 3.9).

Some very important constants are also defined in the script header. These constants are used to identify the registry hive type (lines 61 through 66) and the registry key value type (lines 68 through 72). These constants will be used later in the script.

Before diving into the code, it is important to note that the logic used for the search and replace capabilities is a particular case of the script logic used to browse the registry. We will see this further, but this peculiarity is reflected during the command-line parameter parsing (lines 117 through 135) where the “List,” “Search,” and “Replace” actions define a different browse level value (List=0, Search=1, Replace=2).

→ **Sample 3.3** *Browsing, creating, deleting, searching, and replacing information in the registry
(Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
42:  </runtime>
43:
44:  <script language="VBScript" src=..\Functions\SearchStringFunction.vbs" />
45:  <script language="VBScript" src=..\Functions\ReplaceStringFunction.vbs" />
46:  <script language="VBScript" src=..\Functions\ConvertStringInArrayFunction.vbs" />
47:  <script language="VBScript" src=..\Functions\ConvertArrayInStringFunction.vbs" />
48:  <script language="VBScript" src=..\Functions\TinyErrorHandler.vbs" />
49:
50:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
51:
52:  <script language="VBScript">
53:    <![CDATA[
.:
57:  Const cComputerName = "LocalHost"
58:  Const cWMINameSpace = "Root/default"
59:  Const cWMIClass = "StdRegProv"
60:
61:  Const HKEY_CLASSES_ROOT = &h80000000
62:  Const HKEY_CURRENT_USER = &h80000001
63:  Const HKEY_LOCAL_MACHINE = &h80000002
64:  Const HKEY_USERS = &h80000003
65:  Const HKEY_CURRENT_CONFIG = &h80000005
66:  Const HKEY_DYN_DATA = &h80000006
67:
68:  Const REG_SZ = 1
69:  Const REG_EXPAND_SZ = 2
70:  Const REG_BINARY = 3
71:  Const REG_DWORD = 4
72:  Const REG_MULTI_SZ = 7
73:
74:  Const cBrowse = 0
75:  Const cSearch = 1
76:  Const cReplace = 2
.:
109:  ' -----
110:  ' Parse the command line parameters
111:  If WScript.Arguments.Named.Count = 0 Then
112:    WScript.Arguments.ShowUsage()
113:    WScript.Quit
114:  End If
115:
116:  If WScript.Arguments.Named.Count Then
117:    Select Case Ucase(WScript.Arguments.Named("Action"))
118:      Case "LIST"
119:        boolList = True
120:        intBrowseLevel = cBrowse
121:      Case "SEARCH"
122:        boolList = True
```

```
123:           intBrowseLevel = cSearch
124:           Case "REPLACE"
125:               boolList = True
126:               intBrowseLevel = cReplace
127:           Case "CREATE"
128:               boolCreate = True
129:           Case "DELETE"
130:               boolDelete = True
131:           Case Else
132:               WScript.Echo "Invalid action type. Only [List], [Search], ..." & vbCRLF
133:               WScript.Arguments.ShowUsage()
134:               WScript.Quit
135: End Select
136:
137: strBaseKey = WScript.Arguments.Named("BaseKey")
138: If Len(strBaseKey) = 0 Then
139:     WScript.Echo "Invalid registry key path." & vbCRLF
140:     WScript.Arguments.ShowUsage()
141:     WScript.Quit
142: Else
143:     intDelimiterPosition = InStr (strBaseKey, "\")
144:     If intDelimiterPosition Then
145:         strHiveType = Mid (strBaseKey, 1, intDelimiterPosition - 1)
146:         strBaseKey = Mid (strBaseKey, intDelimiterPosition + 1)
147:     Else
148:         If boolList Then
149:             strHiveType = strBaseKey
150:             strBaseKey = ""
151:         Else
152:             WScript.Echo "Invalid registry key base." & vbCRLF
153:             WScript.Arguments.ShowUsage()
154:             WScript.Quit
155:         End If
156:     End If
157:
158: Select Case Ucase (strHiveType)
159:     Case "HKCLS"
160:         intHiveType = HKEY_CLASSES_ROOT
161:     Case "HKCU"
162:         intHiveType = HKEY_CURRENT_USER
163:     Case "HKLM"
164:         intHiveType = HKEY_LOCAL_MACHINE
165:     Case "HKUSERS"
166:         intHiveType = HKEY_USERS
167:     Case "HKCONFIG"
168:         intHiveType = HKEY_CURRENT_CONFIG
169:     Case "HKDYN"
170:         intHiveType = HKEY_DYN_DATA
171:     Case Else
172:         WScript.Echo "Invalid hive type. Only [HKCLS], [HKCU], ..." & vbCRLF
173:         WScript.Arguments.ShowUsage()
174:         WScript.Quit
175:     End Select
176: End If
177: End If
178:
179: If WScript.Arguments.Unnamed.Count = 1 Then
180:     strKeyName = WScript.Arguments.Unnamed.Item(0)
181:     intDelimiterPosition = InStr (strKeyName, "=")
182:     If intDelimiterPosition Then
```

```
183:         varKeyNameValue = Mid (strKeyName, intDelimiterPosition + 1)
184:         strKeyName = Mid (strKeyName, 1, intDelimiterPosition - 1)
185:     Else
186:         If Not boolDelete = True Then
187:             WScript.Echo "Invalid registry key value." & vbCrLf
188:             WScript.Arguments.ShowUsage()
189:             WScript.Quit
190:         End If
191:     End If
192:     If boolDelete = False Then
193:         Select Case Ucase (WScript.Arguments.Named("KeyType"))
194:             Case "REG_SZ"
195:                 intKeyType = REG_SZ
196:                 boolKeyValue = True
197:             Case "REG_MULTI_SZ"
198:                 varKeyNameValue = ConvertStringInArray (varKeyNameValue, ", ")
199:                 intKeyType = REG_MULTI_SZ
200:                 boolKeyValue = True
201:             Case "REG_EXPAND_SZ"
202:                 intKeyType = REG_EXPAND_SZ
203:                 boolKeyValue = True
204:             Case "REG_BINARY"
205:                 varKeyNameValue = ConvertStringInArray (varKeyNameValue, ", ")
206:                 intKeyType = REG_BINARY
207:                 boolKeyValue = True
208:             Case "REG_DWORD"
209:                 varKeyNameValue = CLng (varKeyNameValue)
210:                 intKeyType = REG_DWORD
211:                 boolKeyValue = True
212:             Case Else
213:                 WScript.Echo "Invalid key type. Only [REG_SZ], ..." & vbCrLf
214:                 WScript.Arguments.ShowUsage()
215:                 WScript.Quit
216:         End Select
217:     End If
218: End If
219:
220: strStringToSearch = WScript.Arguments.Named("SearchString")
221: If Len(strStringToSearch) = 0 And intBrowseLevel = 1 Then
222:     WScript.Echo "Invalid search string."
223:     WScript.Arguments.ShowUsage()
224:     WScript.Quit
225: End If
226:
227: strStringToReplace = WScript.Arguments.Named("ReplaceString")
228: If Len(strStringToSearch) = 0 And Len(strStringToReplace) = 0 And intBrowseLevel = 2 Then
229:     WScript.Echo "Invalid search or replace string."
230:     WScript.Arguments.ShowUsage()
231:     WScript.Quit
232: End If
233:
234: If intBrowseLevel = 0 Then boolDisplayRegValues=WScript.Arguments.Named("DisplayRegValues")
235: If Len(boolDisplayRegValues) = 0 Then boolDisplayRegValues = True
236:
237: strUserID = WScript.Arguments.Named("User")
238: If Len(strUserID) = 0 Then strUserID = ""
239:
240: strPassword = WScript.Arguments.Named("Password")
241: If Len(strPassword) = 0 Then strPassword = ""
242:
```

```

243: strComputerName = WScript.Arguments.Named("Machine")
244: If Len(strComputerName) = 0 Then strComputerName = cComputerName
245:
246: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
247: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
248:
249: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
250:                                         strUserID, strPassword)
...:
253: Set objWMIClass = objWMIServices.Get (cWMIClass)
...:
255:
...:
...:
...:

```

From line 137 through 176, the script parses the `/BaseKey` switch. Basically, it separates the complete registry key path given on the command line into two parts: the registry hive (i.e., HKLM) and the tree path (i.e., SOFTWARE\Microsoft). The registry hive is converted to its corresponding value (lines 158 through 175) based on the constants given in the script header. This value is required for the methods exposed by the `StdRegProv` class.

Lines 179 through 218 parse the registry key name (line 183), its value assignment (line 184), and its type (lines 193 through 216). The logic developed is very similar to the one used to parse the environment variable assignment developed in the previous chapter (see Sample 2.26, “Reading, creating, updating, and deleting environment variables”).

After establishing the WMI connection (lines 249 and 250), the `StdRegProv` class is instantiated. Note that a class instance is created (line 253), since the `StdRegProv` class is defined as a static class (*Static* qualifier).

Based on the command-line parameters, the browse operation is started in Sample 3.4. There is nothing unusual in this piece of code, since the browsing logic is encapsulated in a subfunction (line 261).

Sample 3.4 *Browsing information in the registry (Part II)*

```

...:
...:
...:
255:
256: ' -- LIST -----
257: If boolList = True Then
258:     intCounterValue = 0
259:     intCounterValueName = 0
260:
261:     BrowseRegistry objWMIClass, intHiveType, strBaseKey
262:
263:     Select Case intBrowseLevel
264:         Case cBrowse
265:

```

```
266:         Case csearch
267:             WScript.Echo
268:             WScript.Echo intCounterValue & " registry value(s) found."
269:             WScript.Echo intCounterValueName & " registry value name(s) found."
270:         Case cReplace
271:             WScript.Echo
272:             WScript.Echo intCounterValue & " registry value(s) replaced."
273:             WScript.Echo intCounterValueName & " registry value name(s) replaced."
274:     End Select
275:
276:     WScript.Echo vbCRLF & "Completed."
277:
278: End If
279:
...:
...:
...:
```

Once the browse operation completes, and based on the browse type (simply browsing, searching, or replacing), an output message is displayed to show the number of matches during a search (lines 268 and 269) or a replace (lines 272 and 273). We will examine the `BrowseRegistry()` function in Samples 3.7 and 3.8.

Sample 3.5 *Creating information in the registry (Part III)*

```
...:
...:
...:
279:
280: ' -- CREATE -----
281: If boolCreate = True Then
282:     If boolKeyValue Then
283:         intRC = SetRegistryValue (objWMIClass, intHiveType, _
284:                                 strBaseKey, intKeyType, _
285:                                 strKeyName, varKeyValueName)
286:     If intRC Then
287:         WScript.Echo "Cannot create registry key value '" & strKeyName & _
288:                     "' under '" & strBaseKey & "' (" & intRC & ")."
289:     Else
290:         WScript.Echo "Registry key value '" & strKeyName & "' under '" & _
291:                     strBaseKey & "' created."
292:     End If
293:     Else
294:         intRC = objWMIClass.CreateKey(intHiveType, strBaseKey)
295:         If intRC Then
296:             WScript.Echo "Cannot create registry key '" & strBaseKey & "' (" & intRC & ")."
297:         Else
298:             WScript.Echo "Registry key '" & strBaseKey & "' created."
299:         End If
300:     End If
301: End If
302:
...:
...:
...:
```

To create registry information (see Sample 3.5), two different cases must be considered:

- **The registry tree is already created and a registry key value must be created:** This case corresponds to the logic coded from line 282 through 292. This portion of code sets the registry key value. If the key exists, it is updated; if the key does not exist, it is created. The key value creation or update is encapsulated in the SetRegistryValue() function (line 283). This operation is encapsulated in a subfunction, because a particular *StdRegProv* class method must be invoked based on the registry key type (i.e., REG_SZ, REG_DWORD). This function will also be reused during the replace operation. This case corresponds to the following command-line parameters:

```
C:\>WMIRegistry.wsf MyStringValue=XYW /KeyType:REG_SZ
/BaseKey:HKEY_LOCAL_MACHINE\Software\MyKeys /Action:Create
```

- **The registry tree does not exist and it must be created:** This case corresponds to the logic coded from line 294 through 299. This portion of code creates a registry tree by using the *CreateKey* method exposed by the *StdRegProv* class (line 294). This piece of code is only executed when no registry key assignment is given on the command line (line 282). This case corresponds to the following command-line parameters:

```
C:\>WMIRegistry.wsf /BaseKey:HKEY_LOCAL_MACHINE\Software\MyKeys /Action:Create
```

Sample 3.6 Deleting information in the registry (Part IV)

```
...:
...:
...:
302:
303: ' -- DELETE -----
304: If boolDelete = True Then
305:   If Len (strKeyName) Then
306:     intRC = objWMIClass.DeleteValue (intHiveType, strBaseKey, strKeyName)
307:     If intRC Then
308:       WScript.Echo "Cannot delete registry key value '" & strKeyName & _
309:                   "' under '" & strBaseKey & "' (" & intRC & ")."
310:     Else
311:       WScript.Echo "Registry key value '" & strKeyName & "' under '" & _
312:                   strBaseKey & "' deleted."
313:   End If
314: Else
315:   intRC = objWMIClass.DeleteKey(intHiveType, strBaseKey)
316:   If intRC Then
317:     WScript.Echo "Cannot delete registry key '" & strBaseKey & "' (" & intRC & ")."
318:   Else
319:     WScript.Echo "Registry key '" & strBaseKey & "' deleted."
```

```
320:           End If
321:       End If
322:   End If
...
327:
...
...
...
```

To delete some registry data (see Sample 3.6), two different cases must be considered:

- **A registry value must be deleted:** This case corresponds to the logic coded from line 305 through 313. This portion of code deletes the registry value by using the *DeleteValue* method exposed by the *StdRegProv* class (line 306). This piece of code is only executed when registry key name is given on the command line (line 305). This corresponds to the following command-line parameters:

```
C:\>WMIRegistry.wsf MyStringValue /BaseKey:HKLM\Software\MyKeys /Action:Delete
```

- **A registry key must be deleted:** This case corresponds to the logic coded from line 315 through 321. This portion of code deletes the registry key by using the *DeleteKey* method exposed by the *StdRegProv* class (line 315). Note that it is not possible to delete a complete registry tree with this method. If subkeys exist below the selected registry key, they must be deleted first. This piece of code is only executed when no registry key name is given on the command line (line 305). This case corresponds to the following command-line parameters:

```
C:\>WMIRegistry.wsf /BaseKey:HKLM\Software\MyKeys /Action:Delete
```

When a registry browse operation, search operation, or replace operation is requested, the *BrowseRegistry()* function is invoked (see Sample 3.7). This case corresponds to the following command-line parameters:

```
C:\>WMIRegistry.wsf /BaseKey:HKLM\Software /Action>List /DisplayRegValues-
C:\>WMIRegistry.wsf /BaseKey:HKLM\Software /Action>List /DisplayRegValues+
C:\>WMIRegistry.wsf /BaseKey:HKLM\SOFTWARE\MyKeys /Action:Search /SearchString:String
C:\>WMIRegistry.wsf /BaseKey:HKLM\SOFTWARE\MyKeys /Action:Replace /SearchString:String
/ReplaceString:MyString
```

Displaying registry values occurs by default with a search or replace operation but not with a browse operation. To enable the values to be displayed during a browse, you must supply the */DisplayRegValue+* switch. The *BrowseRegistryValues()* function is invoked to browse the existing registry values (line 340). We will review the *BrowseRegistryValues()* function when discussing Sample 3.8.

Sample 3.7 Browsing, searching, and replacing information in the registry (Part V)

```

...:
...:
...:
327:
328: ' -----
329: Function BrowseRegistry (objWMIClass, intHiveType, strBaseKey)
...:
338:     WScript.Echo strBaseKey
339:     If boolDisplayRegValues Then
340:         BrowseRegistryValues objWMIClass, intHiveType, strBaseKey
341:     End If
342:
343:     If Len (strBaseKey) Then
344:         strBackSlash = "\"
345:     Else
346:         strBackSlash = ""
347:     End If
348:
349:     intRC = objWMIClass.EnumKey (intHiveType, strBaseKey, strSubKeys)
350:     If intRC = 0 Then
351:         If IsNull (strSubKeys) = False Then
352:             If Ubound (strSubKeys) <> -1 Then
353:                 For intIndice = 0 To Ubound (strSubKeys)
354:                     BrowseRegistry objWMIClass, intHiveType,
355:                                 strBaseKey & strBackSlash & strSubKeys (intIndice)
356:                 Next
357:             End If
358:         End If
359:     End If
360:
361: End Function
362:
...:
...:
...:
```

Sample 3.7 retrieves from the base key the list of subkeys available (line 349). The list is returned in an array, called `strSubKeys`, passed as a parameter of the `EnumKey` method of the `StdRegProv` class (line 349). If the array is properly initialized, the `BrowseRegistry()` function is recursively invoked and the new base key is passed by concatenating the current base key with each subkey found in the array (line 355).

If the registry values in a key are browsed, the `BrowseRegistryValues()` function is invoked (line 340). The `BrowseRegistryValues()` function code is shown in Sample 3.8.

Sample 3.8 Browsing, searching, and replacing information in the registry (Part VI)

```
...:  
...:  
...:  
362:  
363: ' -----  
364: Function BrowseRegistryValues (objWMIClass, intHiveType, strKeyPath)  
...:  
383:     intRC = objWMIClass.EnumValues (intHiveType, strKeyPath, strKeyNames, intKeyTypes)  
384:     If intRC = 0 Then  
385:         If IsArray (strKeyNames) Then  
386:             For intIndiceKeyName = 0 To Ubound (strKeyNames)  
387:                 Select Case intKeyTypes (intIndiceKeyName)  
388:                     Case REG_SZ  
389:                         strKeyType = " (REG_SZ) "  
390:                         intRC = objWMIClass.GetStringValue (intHiveType, strKeyPath, _  
391:                                         strKeyNames (intIndiceKeyName), _  
392:                                         varKeyValue)  
393:                     Case REG_EXPAND_SZ  
394:                         strKeyType = " (REG_EXPAND_SZ) "  
395:                         intRC = objWMIClass.GetExpandedStringValue (intHiveType, _  
396:                                         strKeyPath, _  
397:                                         strKeyNames (intIndiceKeyName), _  
398:                                         varKeyValue)  
399:                     Case REG_MULTI_SZ  
400:                         strKeyType = " (REG_MULTI_SZ) "  
401:                         intRC = objWMIClass.GetMultiStringValue (intHiveType, strKeyPath, _  
402:                                         strKeyNames (intIndiceKeyName), _  
403:                                         varKeyValue)  
404:                     Case REG_BINARY  
405:                         strKeyType = " (REG_BINARY) "  
406:                         intRC = objWMIClass.GetBinaryValue (intHiveType, strKeyPath, _  
407:                                         strKeyNames (intIndiceKeyName), _  
408:                                         varKeyValue)  
409:                     Case REG_DWORD  
410:                         strKeyType = " (REG_DWORD) "  
411:                         intRC = objWMIClass.GetDWORDValue (intHiveType, strKeyPath, _  
412:                                         strKeyNames (intIndiceKeyName), _  
413:                                         varKeyValue)  
414:                 End Select  
415:  
416:                 If intRC = 0 Then  
417:                     boolDisplay = False  
418:                     boolDisplayDeleted = False  
419:  
420:                     Select Case intBrowseLevel  
421:                         Case cBrowse  
422:                             boolDisplay = True  
423:                         Case csearch  
424:                             boolFoundInName = SearchString(strKeyNames(intIndiceKeyName),  
425:                                         strStringToSearch)  
426:                             boolFoundInValue = SearchString(varKeyValue, _  
427:                                         strStringToSearch)  
428:                             If boolFoundInValue Then  
429:                                 intCounterValue = intCounterValue + 1  
430:                                 boolDisplay = True  
431:                         End If
```



```
492:                               "=&h" & Hex(varKeyValue)
493:             Case Else
494:                 WScript.Echo strKeyType & _
495:                               strDisplayKeyName & _
496:                               "=" & varKeyValue
497:             End Select
498:
499:             If boolDisplayDeleted Then
500:                 WScript.Echo " Deleting" & strKeyType & _
501:                               strOriginalKeyName & _
502:                               "... "
503:             End If
504:
505:             End If
506:         End If
507:     Next
508: End If
509: End If
510:
511: End Function
512:
...:
...:
...:
```

This function can be divided into three parts:

- **The registry value reading** (lines 383 through 414): This piece of code extracts the collection of values with the *EnumValues* method of the *StdRegProv* class (line 383). This method returns in two arrays the key value name (*strKeyNames*) and its corresponding type (*intKeyTypes*). If the arrays are properly initialized, each array element is examined in a loop (lines 386 through 507) containing the two other portions of code (see the next two bullets). Based on the key value type (i.e., REG_SZ, REG_BINARY), the value is extracted from the registry with the *StdRegProv* method corresponding to the registry value type (lines 387 through 414).
- **The registry value parsing for a search or a replace operation** (lines 416 through 467): Once the value is extracted, the desired browse operation is executed. If it is a simple browse operation to display the registry tree with its key values, the process forces a display of the value by setting a Boolean variable to True (line 422). For a search operation, the code invokes the *SearchString()* function for the examined key name (line 424) with its value (line 426). If a match is found, the matching value will be displayed later and a Boolean variable is set to True (lines 430 and 434). For a replace operation, first the code saves the original key name in a temporary variable (line 437). Next, the code invokes the *ReplaceString()* function for the examined key name (line 442) with its value (line 439). It is impor-

tant to note that the ReplaceString() function automatically replaces the variable content of the key name (“`strKeyNames (intIndiceKeyName)`”) with its value (“`varKeyValue`”) if there is a match. If there is no match, no change to the original content of the variables is made. If a match is found, a new key is created with the updated content of the variables by invoking the SetRegistryValue() function. If the key name is modified by the replace operation, it means that the key with the original name still exists in the registry. This is why the script deletes the original key name by using the temporary variable initialized at line 437. The delete operation is executed at line 459.

- **The registry value display** (lines 469 through 503): To display the key value with its name, the script checks if the key name is not blank. In such a case, it means that the key has no name and corresponds to a default registry value for the key (line 470). Next, the registry key value is converted according to its type (lines 476 through 497) for a suitable display.

Creating or updating a registry key value in the registry is pretty straightforward. The `StdRegProv` method corresponding to the registry key type (i.e., `REG_SZ`, `REG_BINARY`) must be invoked. This logic is shown in Sample 3.9 in lines 521 through 547.

Sample 3.9 *Creating or updating information in the registry (Part VII)*

```
...:
...:
...:
512:
513: '
514: Function SetRegistryValue (objWMIClass, intHiveType, _
515:                               strBaseKey, intKeyType, _
516:                               strKeyName, varKeyNameValue)
...:
521:     Select Case intKeyType
522:         Case REG_SZ
523:             intRC = objWMIClass.SetStringValue (intHiveType, _
524:                                                 strBaseKey, _
525:                                                 strKeyName, _
526:                                                 varKeyNameValue)
527:         Case REG_MULTI_SZ
528:             intRC = objWMIClass.SetMultiStringValue (intHiveType, _
529:                                                 strBaseKey, _
530:                                                 strKeyName, _
531:                                                 varKeyNameValue)
532:         Case REG_EXPAND_SZ
533:             intRC = objWMIClass.SetExpandedStringValue (intHiveType, _
534:                                                 strBaseKey, _
535:                                                 strKeyName, _
536:                                                 varKeyNameValue)
537:         Case REG_BINARY
```

```

538:             intRC = objWMIClass.SetBinaryValue (intHiveType, _
539:                                         strBaseKey, _
540:                                         strKeyName, _
541:                                         varKeyNameValue)
542:             Case REG_DWORD
543:                 intRC = objWMIClass.SetDWORDValue (intHiveType, _
544:                                         strBaseKey, _
545:                                         strKeyName, _
546:                                         varKeyNameValue)
547:             End Select
548:
549:             SetRegistryValue = intRC
550:
551:         End Function
552:
553:     ]]>
554:     </script>
555:     </job>
556:</package>
```

3.3.4 Session providers

The *Session* provider enables the management of network sessions and connections. The provider is implemented as an instance and method provider, as shown in Table 3.9.

Table 3.9 The Session 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
Session Provider	Root\CLIMV2	X	X					X	X	X	X	X	X			

Available in the Root\CLIMV2 namespace, the classes it supports are listed in Table 3.10.

Table 3.10 The Session Provider Classes

Name	Type	Comments
Win32_ServerConnection	Dynamic	Represents the connections made from a remote computer to a shared resource on the local computer.
Win32_ServerSession	Dynamic	Represents the sessions that have been established with the local computer, by users on a remote computer.
Win32_SessionConnection	Association	Represents an association between a session established with the local server, by a user on a remote machine, and the connections that depend on the session.
Win32_ConnectionShare	Association	Relates a shared resource on the computer and the connection made to the shared resource.

On one hand, the *Win32_ServerConnection* is associated with the *Win32_Share* class (with the help of the *Win32_ConnectionShare* association class), and on the other hand the *Win32_ServerConnection* is associated with the *Win32_ServerSession* class (with the help of the *Win32_SessionConnection*). These associations are shown in Figures 3.3 and 3.4, respectively.

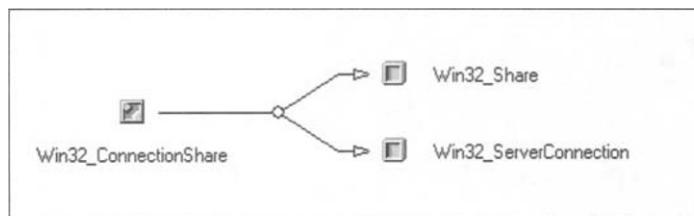


Figure 3.3 The *Win32_Share* and *Win32_ServerConnection* classes are associated with the *Win32_ConnectionShare* association class.

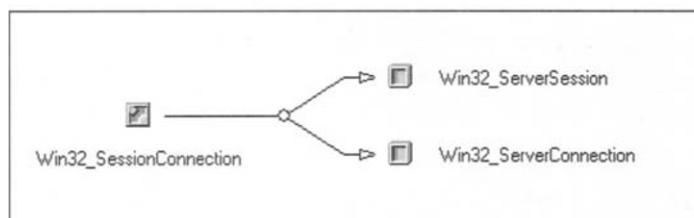


Figure 3.4 The *Win32_ServerSession* and *Win32_ServerConnection* classes are associated with the *Win32_SessionConnection* association class.

The next sample demonstrates the code logic to retrieve the session information with its associated instances. This script sample doesn't expect any switch by default. So, if a machine with the IP address 192.10.10.3 has a connection established by the administrator on a share called "MyShare" and resides on the computer where the script is launched, the output would be as follows:

```

1: C:\>WMISessions.Wsf
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - \\\192.10.10.3\MYSHARE (ADMINISTRATOR) -----
6: ActiveTime: ..... 443
7: *ComputerName: ..... 192.10.10.3
8: ConnectionID: ..... 174
9: InstallDate: ..... 01-01-2000
10: NumberOfFiles: ..... 0
11: NumberOfUsers: ..... 1
12: *ShareName: ..... MYSHARE
13: *UserName: ..... ADMINISTRATOR
  
```

```
14:  
15:    -- Session connection information -----  
16:    ActiveTime: ..... 443  
17:    ClientType: ..... Windows .NET 3718  
18:    *ComputerName: ..... 192.10.10.3  
19:    IdleTime: ..... 8  
20:    InstallDate: ..... 01-01-2000  
21:    ResourcesOpened: ..... 0  
22:    SessionType: ..... 2  
23:    TransportName: ..... \Device\NetbiosSmb  
24:    *UserName: ..... ADMINISTRATOR  
25:  
26:    -- Share connection information -----  
27:    AllowMaximum: ..... TRUE  
28:    Caption: ..... MYSHARE  
29:    Description: .....  
30:    InstallDate: ..... 01-01-2000  
31:    *Name: ..... MYSHARE  
32:    Path: ..... J:\MYDIR  
33:    Status: ..... OK  
34:    Type: ..... Disk
```

First, the script retrieves the *Win32_ServerConnection* instances (lines 5 through 13); next, it retrieves the *Win32_ServerSession* associated instances (lines 15 through 24) with some basic information about the *Win32_Share* associated instance (lines 26 and 27).

It is also possible to delete a session based on the computer name. The command line will be as follows:

```
C:\>WMISessions.Wsf /Delete:192.10.10.3  
Microsoft (R) Windows Script Host Version 5.6  
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
Session '192.10.10.3' deleted.  
Completed.
```

The code is shown in Sample 3.10. Since the command-line parameters definition and parsing are pretty simple and use the same logic as previous samples, they have been skipped. Right after the WMI connection (lines 69 through 73), the script checks if a computer name is specified with the */Delete* switch. If not, it processes the display of the session information (lines 78 through 150). If a computer name is specified, it processes the deletion of the sessions established from the given computer name (lines 153 through 167).

Sample 3.10 *Viewing the active sessions with their associations*

```
1:<?xml version="1.0"?>  
.:  
8:<package>  
9:  <job>  
...:  
13:    <runtime>
```

```
...  
24: </runtime>  
25:  
26: <script language="VBScript" src="..\Functions\DecodeShareTypeFunction.vbs" />  
27: <script language="VBScript" src="..\Functions\DisplayFormattedPropertyFunction.vbs" />  
28: <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />  
29:  
30: <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>  
31: <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />  
32:  
33: <script language="VBScript">  
34: <![CDATA[  
...  
38: Const cComputerName = "LocalHost"  
39: Const cWMINameSpace = "Root/cimv2"  
40: Const cWMIClass = "Win32_ServerConnection"  
41: Const cWMISSessionClass = "Win32_ServerSession"  
...  
56: ' -----  
...  
69: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault  
70: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate  
71:  
72: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _  
73: strUserID, strPassword)  
...  
76: If Len(strComputer) = 0 Then  
77:     ' -- LIST -----  
78:     Set objWMIInstances = objWMIServices.InstancesOf (cWMIClass)  
...  
81:     If objWMIInstances.Count Then  
82:         For Each objWMIInstance In objWMIInstances  
83:             WScript.Echo "- \" & Ucase (objWMIInstance.ComputerName) & _  
84:                 "\" & Ucase (objWMIInstance.ShareName) & _  
85:                 " (" & Ucase (objWMIInstance.UserName) & _  
86:                 ") " & String (60, "-")  
87:             Set objWMIPropertySet = objWMIInstance.Properties_  
88:             For Each objWMIProperty In objWMIPropertySet  
89:                 DisplayFormattedProperty objWMIInstance, _  
90:                     objWMIProperty.Name, _  
91:                     objWMIProperty.Name, _  
92:                     Null  
93:             Next  
...  
96:             Set objWMIAssocInstances = objWMIServices.ExecQuery _  
97:                 ("Associators of (" & objWMIInstance.Path_.RelPath & _  
98:                 ") Where AssocClass=Win32_SessionConnection")  
...  
101:            If objWMIAssocInstances.Count Then  
102:                WScript.Echo  
103:                WScript.Echo " -- Session connection information " & " " & String (53, "-")  
104:                For Each objWMIAssocInstance In objWMIAssocInstances  
105:                    Set objWMIPropertySet = objWMIAssocInstance.Properties_  
106:                    For Each objWMIProperty In objWMIPropertySet  
107:                        DisplayFormattedProperty objWMIAssocInstance, _  
108:                            " " & objWMIProperty.Name, _  
109:                            objWMIProperty.Name, _  
110:                            Null  
111:                Next  
112:                Set objWMIPropertySet = Nothing
```

```
113:           Next
114:       End If
115:
116:       Set objWMIAssocInstances = objWMIServices.ExecQuery _
117:             ("Associators of {" & objWMIInstance.Path_.RelPath & _
118:             "} Where AssocClass=Win32_ConnectionShare")
...
121:       If objWMIAssocInstances.Count Then
122:           WScript.Echo
123:           WScript.Echo " -- Share connection information " & " " & String (53, "-")
124:           For Each objWMIAssocInstance In objWMIAssocInstances
125:               Set objWMIPropertySet = objWMIAssocInstance.Properties_
126:               For Each objWMIProperty In objWMIPropertySet
127:                   Select Case objWMIProperty.Name
128:                       Case "Type"
129:                           DisplayFormattedProperty objWMIAssocInstance, _
130:                             " " & objWMIProperty.Name, _
131:                             ShareType (objWMIProperty.Value), _
132:                             Null
133:
134:                       Case Else
135:                           DisplayFormattedProperty objWMIAssocInstance, _
136:                             " " & objWMIProperty.Name, _
137:                             objWMIProperty.Name, _
138:                             Null
139:                   End Select
140:               Next
141:               Set objWMIPropertySet = Nothing
142:           Next
143:       End If
144:       WScript.Echo
...
147:       Next
148:   Else
149:       WScript.Echo "No session." & vbCRLF
150:   End If
151: Else
152:     '-- Delete -----
153:     Set objWMIInstances = objWMIServices.ExecQuery ("Select * From " & cWMISessionClass & _
154:                                         " Where ComputerName=' " & strComputer & "'")
...
157:     If objWMIInstances.Count Then
158:         For Each objWMIInstance In objWMIInstances
159:             objWMIInstance.Delete_
...
162:         WScript.Echo "Session '" & objWMIInstance.ComputerName & "' deleted."
163:         Next
164:         WScript.Echo
165:     End If
...
168: End If
169:
170: WScript.Echo "Completed."
...
176: ]]>
177: </script>
178: </job>
179:</package>
```

When the script displays session information, it retrieves a collection of *Win32_ServerConnection* instances (line 78). For each instance (lines 82 through 147), the script displays the instance properties (lines 87 through 93) with the existing associated instances. As we have seen in Figures 3.3 and 3.4, two associated instances with their properties can be retrieved:

- The *Win32_ServerSession* instances associated with the *Win32_SessionConnection* association class (lines 96 through 114)
- The *Win32_Share* instances associated with the *Win32_ConnectionShare* association class (lines 116 through 143).

Once completed, the script terminates its execution.

With the /Delete switch, sessions established by a specific computer can be deleted (lines 153 through 167). Instead of retrieving a specific session, based on the command-line parameters, the script retrieves all sessions available for the given computer (lines 153 and 154) and deletes them (line 159) in a loop (lines 158 through 163).

3.3.5 Kernel Job providers

Installed on Windows XP and Windows Server 2003, the *Kernel Job Object* provider enables access to data on named kernel job objects. This provider does not report unnamed kernel job objects. The *Kernel Job Object* providers capabilities are listed in Table 3.11.

Table 3.11

The Kernel Job Object Providers 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
Job Object Providers																
NamedJobObjectActgInfoProv	Root/CIMv2	X						X	X	X	X	X	X			
NamedJobObjectLimitSettingProv	Root/CIMv2	X						X	X	X	X	X	X			
NamedJobObjectProv	Root/CIMv2	X						X	X	X	X	X	X			
NamedJobObjectSecLimitSettingProv	Root/CIMv2	X						X	X	X	X	X	X			

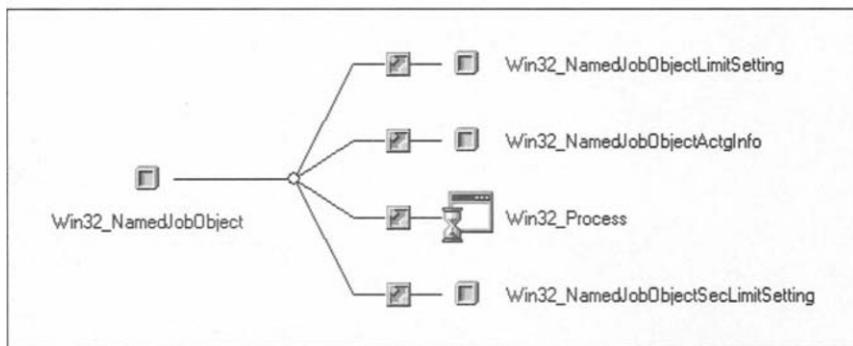
The classes supported by these providers are available in the Root\CMV2 namespace and summarized in Table 3.12.

→ **Table 3.12** *The Kernel Job Object Classes*

Name	Type	Comments
Win32_NamedJobObject	Dynamic	Represents a kernel object that is used to group processes for the sake of controlling the life and resources of the processes within the job object.
Win32_NamedJobObjectActglInfo	Dynamic	Represents the I/O accounting information for a job object.
Win32_NamedJobObjectLimitSetting	Dynamic	Represents the limit settings for a job object.
Win32_NamedJobObjectSecLimitSetting	Dynamic	Represents the security limit settings for a job object.
Win32_SIDandAttributes	Dynamic	Represents a security identifier (SID) and its attributes.
Win32_NamedJobObjectProcess	Association Aggregation	Relates a job object and the process contained in the job object.
Win32_CollectionStatistics	Association	Relates a managed system element collection and the class representing statistical information about the collection.
Win32_NamedJobObjectLimit	Association	Represents an association between a job object and the job object limit settings.
Win32_NamedJobObjectSecLimit	Association	Relates a job object and the job object security limit settings.
Win32_NamedJobObjectStatistics	Association	Represents an association between a job object and the job object I/O accounting information class.
Win32_LUID	Abstract	Represents a locally unique identifier (LUID).
Win32_LUIDandAttributes	Abstract	Represents a LUID and its attributes.
Win32_TokenGroups	Abstract	Represents information about the group SIDs in an access token.
Win32_TokenPrivileges	Abstract	Represents information about a set of privileges for an access token.

The *Win32_NamedJobObject* class represents a kernel object that is used to group processes for the sake of controlling the life and resources of the processes within the job object. Figure 3.5 shows the associations in place.

→ **Figure 3.5**
The Win32_NamedJobObject associations.



As we can see in Figure 3.5, the *Win32_NamedJobObject* has an association with the *Win32_Process*. In the previous chapter, we developed a script using the *Win32_Process* (see Sample 2.50, “Viewing, creating, and killing processes”). We can easily extend this previous sample to show the information available from the associations shown in Figure 3.5. We won’t review the complete script. Only the portion of the code showing the process information will be examined. Until line 333, the script is exactly the same as Sample 2.50. The portion of code is included from line 335 through 389 and is executed for each *Win32_Process* instance examined during the loop (lines 289 through 386).

Sample 3.11 Viewing Job kernel instance associated with a process

```
...:  
...:  
...:  
275: ' - LIST -----  
276: If boolList Then  
277:     If Len (strExecutable) Then  
278:         Set objWMIProcInstances = objWMIServices.ExecQuery ("Select * From " & _  
279:                                         cWMIProcClass & _  
280:                                         " Where Name=' " & _  
281:                                         strExecutable & "'")  
282:         If Err.Number Then ErrorHandler (Err)  
283:     Else  
284:         Set objWMIProcInstances = objWMIServices.InstancesOf (cWMIProcClass)  
285:         If Err.Number Then ErrorHandler (Err)  
286:     End If  
287:  
288:     If objWMIProcInstances.Count Then  
289:         For Each objWMIProcInstance in objWMIProcInstances  
290:             WScript.Echo " - " & Ucase(objWMIProcInstance.Name) & String (60, "-")  
291:             Set objWMIPropertySet = objWMIProcInstance.Properties_  
292:             For Each objWMIProperty In objWMIPropertySet  
293:                 Select Case objWMIProperty.Name  
294:                     Case "Caption"  
295:                         Case "Description"  
296:                         Case "Name"  
297:                         Case "CSCreationClassName"  
298:                         Case "OSCreationClassName"  
299:                         Case "CreationClassName"  
300:                         Case Else  
301:                             DisplayFormattedProperty objWMIProcInstance, _  
302:                                         " " & objWMIProperty.Name, _  
303:                                         objWMIProperty.Name, _  
304:                                         Null  
305:                         End Select  
306:             Next  
307:             Set objWMIPropertySet = Nothing  
308:         WScript.Echo  
309:  
310:         intRC = objWMIProcInstance.GetOwner (strOwnerUser, strOwnerDomain)  
311:         If intRC = 0 Then  
312:             DisplayFormattedProperty objWMIProcInstance, _  
313:                                         " Process owner", _  
314:                                         strOwnerDomain, _  
315:                                         strOwnerUser  
316:         End If  
317:         intRC = objWMIProcInstance.GetOwnerSid (strOwnerSID)  
318:         If intRC = 0 Then  
319:             DisplayFormattedProperty objWMIProcInstance, _  
320:                                         " Process owner SID", _  
321:                                         strOwnerSID, _  
322:                                         Null  
323:         End If  
324:     End If
```

```
332:                               WScript.Echo
333:
334:
335:      Set objWMIJobInstances = objWMIServices.ExecQuery _
336:                                ("Associators of {" &
337:                                objWMIProcInstance.Path_.RelPath & -
338:                                ") Where AssocClass=" & -
339:                                "Win32_NamedJobObjectProcess")
340:
341:      If objWMIJobInstances.Count Then
342:          For Each objWMIJobInstance in objWMIJobInstances
343:              WScript.Echo " - " & Ucase(objWMIJobInstance.CollectionID) & -
344:                          " " & String (60, "-")
345:              Set objWMIPropertySet = objWMIJobInstance.Properties_
346:              For Each objWMIProperty In objWMIPropertySet
347:                  DisplayFormattedProperty objWMIJobInstance, -
348:                                  " " & objWMIProperty.Name, -
349:                                  objWMIProperty.Name, -
350:                                  Null
351:
352:              Next
353:              Set objWMIPropertySet = Nothing
354:
355:
356:              For Each strAssocClass In Array ("Win32_NamedJobObjectLimit", -
357:                                              "Win32_NamedJobObjectSecLimit", -
358:                                              "Win32_NamedJobObjectStatistics")
359:
360:                  Set objWMIJobAssocInstances = objWMIServices.ExecQuery -
361:                                ("Associators of {" &
362:                                objWMIJobInstance.Path_.RelPath & -
363:                                ") Where AssocClass=" & -
364:                                strAssocClass)
365:
366:                  If objWMIJobAssocInstances.Count Then
367:                      For Each objWMIJobAssocInstance in objWMIJobAssocInstances
368:                          WScript.Echo " - " & _
369:                                      Ucase(objWMIJobAssocInstance.Path_.Class) & -
370:                                      " " & String (60, "-")
371:                          Set objWMIPropertySet = objWMIJobAssocInstance.Properties_
372:                          For Each objWMIProperty In objWMIPropertySet
373:                              DisplayFormattedProperty objWMIJobAssocInstance, -
374:                                  " " & objWMIProperty.Name, -
375:                                  objWMIProperty.Name, -
376:                                  Null
377:
378:                      Next
379:                      Set objWMIPropertySet = Nothing
380:
381:                      WScript.Echo
382:                      Next
383:                  End If
384:              Next
385:          End If
386:      Next
387:  Else
388:      WScript.Echo "No information available."
389: End If
...:
...:
...:
```

At line 335, the script retrieves the *Win32_NamedJobObject* instances associated with the *Win32_NamedObjectProcess* association class. If instances of the *Win32_NamedJobObject* class are available (line 341), the script displays their properties (lines 345 through 352). Next, for each instance of the *Win32_NamedJobObject* class, the script retrieves three associated instances (lines 356 through 383):

- The *Win32_NamedJobObjectLimitSetting* instances with the *Win32_NamedJobObjectLimit* association class.
- The *Win32_NamedJobObjectActgInfo* instances with the *Win32_NamedJobObjectActgInfo* association class.
- The *Win32_NamedJobObjectSecLimitSetting* instances with the *Win32_NamedJobObjectSecLimitSetting* association class.

The obtained output for the WMI process is as follows:

```
1: C:\>WMIProcess /Action>List /Executable:wmiprvse.exe
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: ~ WMIPRVSE.EXE-----
6: CreationDate: ..... 09-02-2002 07:18:37
7: CSName: ..... NET-DPEN6400A
8: *Handle: ..... 2712
9: HandleCount: ..... 175
10: InstallDate: ..... 09-02-2002 07:18:37
11: KernelModeTime: ..... 24234848
12: OSName: ..... Microsoft Windows .NET Enterprise Serv...
13: OtherOperationCount: ..... 4967
14: OtherTransferCount: ..... 44708
15: PageFaults: ..... 10974
16: PageFileUsage: ..... 3674112
17: ParentProcessId: ..... 696
18: PeakPageFileUsage: ..... 3977216
19: PeakVirtualSize: ..... 29556736
20: PeakWorkingSetSize: ..... 7913472
21: Priority: ..... 8
22: PrivatePageCount: ..... 3674112
23: ProcessId: ..... 2712
24: QuotaNonPagedPoolUsage: ..... 5656
25: QuotaPagedPoolUsage: ..... 25264
26: QuotaPeakNonPagedPoolUsage: ..... 7744
27: QuotaPeakPagedPoolUsage: ..... 26268
28: ReadOperationCount: ..... 2846
29: ReadTransferCount: ..... 938974
30: SessionId: ..... 0
31: TerminationDate: ..... 09-02-2002 07:18:37
32: ThreadCount: ..... 9
33: UserModeTime: ..... 60887552
34: VirtualSize: ..... 27336704
35: WindowsVersion: ..... 5.1.3590
36: WorkingSetSize: ..... 7864320
37: WriteOperationCount: ..... 2383
```

```
38: WriteTransferCount: ..... 144244
39:
40: Process owner: ..... NT AUTHORITY NETWORK SERVICE
41: Process owner SID: ..... S-1-5-20
42:
43: - \WMI\PROVIDER\SUB\SYSTEM\HOST\JOB -----
44: BasicUIRestrictions: ..... 0
45: *CollectionID: ..... \wmi\provider\sub\system\host\job
46:
47: - WIN32_NAMEDJOBOBJECTLIMITSETTING -----
48: ActiveProcessLimit: ..... 32
49: Affinity: ..... 0
50: JobMemoryLimit: ..... 1073741824
51: LimitFlags: ..... 11016
52: MaximumWorkingSetSize: ..... 0
53: MinimumWorkingSetSize: ..... 0
54: PerJobUserTimeLimit: ..... 0
55: PerProcessUserTimeLimit: ..... 0
56: PriorityClass: ..... 32
57: ProcessMemoryLimit: ..... 134217728
58: SchedulingClass: ..... 5
59: *SettingID: ..... \wmi\provider\sub\system\host\job
60:
61: - WIN32_NAMEDJOBOBJECTSECLIMITSETTING -----
62: PrivilegesToDelete: ..... <OBJECT>
63: RestrictedSIDs: ..... <OBJECT>
64: SecurityLimitFlags: ..... 0
65: *SettingID: ..... \wmi\provider\sub\system\host\job
66: SIDsToDelete: ..... <OBJECT>
67:
68: - WIN32_NAMEDJOBOBJECTACTGINFO -----
69: ActiveProcesses: ..... 1
70: *Name: ..... \wmi\provider\sub\system\host\job
71: OtherOperationCount: ..... 4975
72: OtherTransferCount: ..... 44708
73: PeakJobMemoryUsed: ..... 3977216
74: PeakProcessMemoryUsed: ..... 4136960
75: ReadOperationCount: ..... 2885
76: ReadTransferCount: ..... 941298
77: ThisPeriodTotalKernelTime: ..... 24935856
78: ThisPeriodTotalUserTime: ..... 62790288
79: TotalKernelTime: ..... 24935856
80: TotalPageFaultCount: ..... 11105
81: TotalProcesses: ..... 1
82: TotalTerminatedProcesses: ..... 0
83: TotalUserTime: ..... 62790288
84: WriteOperationCount: ..... 2422
85: WriteTransferCount: ..... 146912
```

3.3.6 TrustMon provider

Only available under Windows Server 2003, the *TrustMon* provider is an instance provider that only supports the “Get” and “Enumeration” operations (Table 3.13).

→ **Table 3.13** *The TrustMon 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
Trust Monitoring Providers	Root\ActiveDirectory	X						X	X		X					

Supporting three classes available in the Root\MicrosoftActiveDirectory namespace (Table 3.14), the provider purpose is to verify the state of the existing trust relationships between domains.

→ **Table 3.14** *The TrustMon Provider Classes*

Name	Type	Comments
Microsoft_TrustProvider	Dynamic (Singleton)	Provider parameterization class.
Microsoft_DomainTrustStatus	Dynamic	Trust enumerator class.
Microsoft_LocalDomainInfo	Dynamic (Singleton)	Local domain information class.

The *Microsoft_TrustProvider* class is a singleton class and includes properties that control how domains will be enumerated with the *Microsoft_DomainTrustStatus* class. The *TrustMon* provider caches the last trust enumeration (default *ListLifeTime* = 20 min) and the last request for the trust status (default *StatusLifeTime* = 3 min). It is also possible to define the trust check level (Table 3.15), if only the trusting domains must be returned (without the trusted domains).

→ **Table 3.15** *The TrustMon Check Levels*

Meaning	Values
Enumerate only	0
Enumerate with SC_QUERY	1
Enumerate with password check	2
Enumerate with SC_RESET	3

The *Microsoft_LocalDomainInfo* class is used to gather information about the domain, while the last class, the *Microsoft_DomainTrustStatus* class, is used to enumerate the trust status. The verification of the trusts is made during the enumeration of the *Microsoft_DomainTrustStatus* class based on the parameters given at the level of the *Microsoft_TrustProvider* instance. It is important to note that the effectiveness of the updated

parameters is not immediate, even if the *Microsoft_TrustProvider* instance is updated immediately before the trust enumeration. The only way to immediately change the provider parameters is to update the settings stored in the repository and stopping/restarting the WMI service **WinMgmt.Exe**. Based on this information, Sample 3.12 has the following command-line parameters:

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

Usage: WMITrust.wsf /Check:value [/ListLifeTime:value] [/StatusLifeTime:value] [/TrustingOnly[+|-]]
          [/DomainInfo[+|-]] [/Machine:value] [/User:value] [/Password:value]

Options:

Check      : Specify the verification to perform: [EnumOnly], [EnumQuery],
              [EnumPwdCheck] or [EnumReset].
ListLifeTime : Time in minutes to cache the last trust enumeration.
StatusLifeTime : Time in minutes to cache the last request for status.
TrustingOnly : If TRUE, enumerations return trusting as well as trusted domains.
DomainInfo   : Provide information about the domain on which this instance of
              the trust monitor is running.
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)

Examples:

WMITrust /DomainInfo+
WMITrust /Check:EnumOnly /StatusLifeTime:3 /ListLifeTime:20 /TrustingOnly+
WMITrust /Check:EnumQuery /ListLifeTime:20 /TrustingOnly+
WMITrust /Check:EnumPwdCheck /StatusLifeTime:3 /TrustingOnly+
WMITrust /Check:EnumReset /TrustingOnly+
```

Requesting the domain information will also show the *TrustMon* provider parameters. For example, the following command line would give:

```
C:\>WMI /DomainInfo+
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

- Local Domain information -----
DCname: ..... NET-DPEN6400A
DNSname: ..... LissWare.NET
FlatName: ..... LISSWARENET
SID: ..... S-1-5-21-1935655697-839522115-1708537768
TreeName: ..... LissWare.Net

- Trust provider parameters -----
ReturnAll: ..... TRUE
TrustCheckLevel: ..... 2
TrustListLifetime: ..... 0
TrustStatusLifetime: ..... 0
```

While the following command line would give:

```
C:\>WMITrust /Check:EnumPwdCheck /TrustingOnly-
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

FlatName: ..... EMEA
SID: ..... S-1-5-21-1708537768-854245398-1957994488
TrustAttributes: ..... 0
TrustDirection: ..... Bi-directional
TrustedDCName: .....
TrustedDomain: ..... Emea.LissWare.Net
TrustIsOk: ..... False
TrustStatus: ..... 1355
TrustStatusString: ..... The specified domain either does not exist
or could not be contacted.
TrustType: ..... Uplevel

FlatName: ..... MYNT40DOMAIN
SID: ..... S-1-5-21-42165204-196285673-1159422225
TrustAttributes: ..... 0
TrustDirection: ..... Inbound
TrustedDCName: .....
TrustedDomain: ..... HOME
TrustIsOk: ..... True
TrustStatus: ..... 0
TrustStatusString: ..... Inbound-only trusts are verified from the trusting side.
TrustType: ..... Downlevel
```

Sample 3.12 shows the code logic developed on top of the *TrustMon* provider. The command-line parameter definitions and parsing follow the same structure as previous script samples.

Sample 3.12 Verifying trusts

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
31:   </runtime>
32:
33:   <script language="VBScript" src=".\\Functions\\DecodeTrustsFunction.vbs" />
34:
35:   <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
36:   <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
37:
38:   <object progid="WbemScripting.SWbemLocator" id="objWMLocator" reference="true"/>
39:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
40:
41:   <script language="VBscript">
42:     <![CDATA[
.:
46:     Const cComputerName = "LocalHost"
47:     Const cWMINameSpace = "Root\\MicrosoftActiveDirectory"
```

```
48: Const cWMITrustProvClass = "Microsoft_TrustProvider"
49: Const cWMIDomTrustClass = "Microsoft_DomainTrustStatus"
50: Const cWMIDomInfoClass = "Microsoft_LocalDomainInfo"
...
72: '
73: ' Parse the command line parameters
74: If WScript.Arguments.Named.Count = 0 Then
75:     WScript.Arguments.ShowUsage()
76:     WScript.Quit
77: End If
...
122: If Len(strComputerName) = 0 Then strComputerName = cComputerName
123:
124: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
125: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
126:
127: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINamespace,
128:                                                 strUserID, strPassword)
...
131: If boolDomainInfo = True Then
132: '
133: WScript.Echo "- Local Domain information " & String (60, "-")
134:
135: Set objWMIInstance = objWMIServices.Get (cWMIDomInfoClass & "=@")
136: If Err.Number Then ErrorHandler (Err)
137:
138: Set objWMIPropertySet = objWMIInstance.Properties_
139: For Each objWMIProperty In objWMIPropertySet
140:     DisplayFormattedProperty objWMIInstance,
141:             " " & objWMIProperty.Name, _
142:             objWMIProperty.Name, _
143:             Null
144: Next
145:
146: WScript.Echo
147: WScript.Echo "- Trust provider parameters " & String (60, "-")
148:
149: Set objWMIInstance = objWMIServices.Get (cWMITrustProvClass & "=@")
...
152: Set objWMIPropertySet = objWMIInstance.Properties_
153: For Each objWMIProperty In objWMIPropertySet
154:     DisplayFormattedProperty objWMIInstance,
155:             " " & objWMIProperty.Name, _
156:             objWMIProperty.Name, _
157:             Null
158: Next
...
162: Else
163: '
164: Set objWMIInstance = objWMIServices.Get (cWMITrustProvClass & "=@")
...
167: objWMIInstance.TrustCheckLevel = intCheckLevel
168:
169: If intStatusLifeTime <> - 1 Then
170:     objWMIInstance.TrustStatusLifetime = intStatusLifeTime
171: End If
172: If intListLifeTime <> - 1 Then
173:     objWMIInstance.TrustListLifetime = intListLifeTime
174: End If
175: If Len(boolTrustingOnly) Then
```

```

176:         objWMIInstance.ReturnAll = Not boolTrustingOnly
177:     End If
178:
179:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete)
...:
184:     ' -----
185:     Set objWMInstances = objWMIServices.InstancesOf (cWMIDomTrustClass)
...:
188:     For Each objWMIInstance In objWMInstances
189:         Set objWMIPropertySet = objWMIInstance.Properties_
190:         For Each objWMIProperty In objWMIPropertySet
191:             Select Case objWMIProperty.Name
192:                 Case "TrustDirection"
193:                     DisplayFormattedProperty objWMIInstance, _
194:                         " " & objWMIProperty.Name, _
195:                         DecodeTrustDirection (objWMIProperty.Value), _
196:                         Null
197:                 Case "TrustType"
198:                     DisplayFormattedProperty objWMIInstance, _
199:                         " " & objWMIProperty.Name, _
200:                         DecodeTrustType (objWMIProperty.Value), _
201:                         Null
202:                 Case "TrustAttributes"
203:                     DisplayFormattedProperty objWMIInstance, _
204:                         " " & objWMIProperty.Name, _
205:                         DecodeTrustAttributes (objWMIProperty.Value), _
206:                         Null
207:                 Case Else
208:                     DisplayFormattedProperty objWMIInstance, _
209:                         " " & objWMIProperty.Name, _
210:                         objWMIProperty.Value, _
211:                         Null
212:             End Select
213:         Next
...:
216:     Next
...:
219: End If
...:
223: ]]>
224: </script>
225: </job>
226:</package>
```

When the **/DomainInfo+** switch is supplied on the command line, the script gets the *Microsoft_LocalDomainInfo* (lines 135 through 144) and *Microsoft_TrustProvider* instances (lines 149 through 158) to display their properties. If the **/DomainInfo** switch is not supplied, the script updates the *Microsoft_TrustProvider* instance with the command-line parameters (lines 164 through 179) and enumerates the list of trusts in place (lines 185 through 216).

To detect a trust status change, a WQL event query can be used. Since the *TrustMon* provider is not implemented as an event provider, the WQL event query must use the **WITHIN** statement. For example, a valid WQL would be as follows:

```
1:  C:\>GenericEventAsyncConsumer.wsf "Select * FROM __InstanceModificationEvent Within 5
   Where TargetInstance ISA 'Microsoft_DomainTrustStatus'
   /NameSpace:Root\MicrosoftActiveDirectory
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Monday, 10 June, 2002 at 12:11:12: '__InstanceModificationEvent' has been triggered.
9: PreviousInstance (wbemCimtypeObject)
10: FlatName (wbemCimtypeString) = EMEA
11: SID (wbemCimtypeString) = S-1-5-21-1060284298-484763869-1343024091
12: TrustAttributes (wbemCimtypeUInt32) = 0
13: TrustDirection (wbemCimtypeUInt32) = 3
14: TrustedDCName (wbemCimtypeString) = \\net-dpep6400a.Emea.LissWare.NET
15: *TrustedDomain (wbemCimtypeString) = Emea.LissWare.NET
16: TrustIsOk (wbemCimtypeBoolean) = True
17: TrustStatus (wbemCimtypeUInt32) = 0
18: TrustStatusString (wbemCimtypeString) = The secure channel was reset and the trust is OK.
19: TrustType (wbemCimtypeUInt32) = 2
20: SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
21: TargetInstance (wbemCimtypeObject)
22: FlatName (wbemCimtypeString) = EMEA
23: SID (wbemCimtypeString) = S-1-5-21-1060284298-484763869-1343024091
24: TrustAttributes (wbemCimtypeUInt32) = 0
25: TrustDirection (wbemCimtypeUInt32) = 3
26: TrustedDCName (wbemCimtypeString) = \\net-dpep6400a.Emea.LissWare.NET
27: *TrustedDomain (wbemCimtypeString) = Emea.LissWare.NET
28: TrustIsOk (wbemCimtypeBoolean) = False
29: TrustStatus (wbemCimtypeUInt32) = 1311
30: TrustStatusString (wbemCimtypeString) = There are currently no logon servers
   available to service the logon request.
31: TrustType (wbemCimtypeUInt32) = 2
32: TIME_CREATED (wbemCimtypeUInt64) = (null)
33:
34: END - OnObjectReady.
35:
36: BEGIN - OnObjectReady.
37: Monday, 10 June, 2002 at 12:11:12: '__InstanceModificationEvent' has been triggered.
38: PreviousInstance (wbemCimtypeObject)
39: FlatName (wbemCimtypeString) = EMEA
40: SID (wbemCimtypeString) = S-1-5-21-1060284298-484763869-1343024091
41: TrustAttributes (wbemCimtypeUInt32) = 0
42: TrustDirection (wbemCimtypeUInt32) = 3
43: TrustedDCName (wbemCimtypeString) = \\net-dpep6400a.Emea.LissWare.NET
44: *TrustedDomain (wbemCimtypeString) = Emea.LissWare.NET
45: TrustIsOk (wbemCimtypeBoolean) = False
46: TrustStatus (wbemCimtypeUInt32) = 1311
47: TrustStatusString (wbemCimtypeString) = There are currently no logon servers
   available to service the logon request.
48: TrustType (wbemCimtypeUInt32) = 2
49: SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
50: TargetInstance (wbemCimtypeObject)
51: FlatName (wbemCimtypeString) = EMEA
52: SID (wbemCimtypeString) = S-1-5-21-1060284298-484763869-1343024091
53: TrustAttributes (wbemCimtypeUInt32) = 0
54: TrustDirection (wbemCimtypeUInt32) = 3
55: TrustedDCName (wbemCimtypeString) = \\net-dpep6400a.Emea.LissWare.NET
56: *TrustedDomain (wbemCimtypeString) = Emea.LissWare.NET
```

```

57:     TrustIsOk (wbemCimtypeBoolean) = False
58:     TrustStatus (wbemCimtypeUInt32) = 1355
59:     TrustStatusString (wbemCimtypeString) = The specified domain either does not exist
      or could not be contacted.
60:     TrustType (wbemCimtypeUInt32) = 2
61:     TIME_CREATED (wbemCimtypeUInt64) = (null)
62:
63: END - OnObjectReady.

```

From line 7 through 34, the trust status changes from 0 to 1311 (line 17 versus line 29), generating an error message at line 30. From line 36 through 63, the trust status changes from 1311 to 1355 (line 46 versus line 58), generating a different error message at line 59.

3.3.7 Windows Proxy provider

The *Windows Proxy* provider supports one single class called *Win32_Proxy* and is located in the **Root\CIMv2** namespace. This provider is implemented as an instance and a method provider (Table 3.16) and manages the Windows Proxy LAN settings.

→ **Table 3.16** *The Windows Proxy Providers 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
Proxy Provider																
Win32_WIN32_PROXY_Prov	Root/CIMv2	X	X					X	X	X	X	X	X	X		

The information exposed by the *Win32_Proxy* class is similar to the information exposed by the *MicrosoftIE_LanSettings* class in **Root\CIMV2\ Applications\MicrosoftIE** namespace. However, the *MicrosoftIE_LanSettings* class uses the *IEINFO5* provider. With the *SetProxySetting* method exposed by the *Win32_Proxy* class, it is possible to update the Proxy LAN settings. Sample 3.13 shows how to proceed.

→ **Sample 3.13** *Managing the Windows Proxy LAN settings*

```

1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:

```

```
26:    </runtime>
27:
28:    <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
29:    <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
30:
31:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
32:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
33:
34:    <script language="VBScript">
35:        <![CDATA[
36:            ...
37:
38:            Const cComputerName = "LocalHost"
39:            Const cWMINameSpace = "Root\\CIMv2"
40:            Const cWMIClass = "Win32_Proxy"
41:
42:            ' -----
43:
44:            ' Parse the command line parameters
45:            If WScript.Arguments.Named.Count = 0 Then
46:                WScript.Arguments.ShowUsage()
47:                WScript.Quit
48:            End If
49:
50:
51:            strComputerName = WScript.Arguments.Named("Machine")
52:            If Len(strComputerName) = 0 Then strComputerName = cComputerName
53:
54:
55:            objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
56:            objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
57:
58:            Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
59:                strUserID, strPassword)
60:
61:
62:            Set objWMIInstances = objWMIServices.InstancesOf (cWMIClass)
63:
64:
65:            If objWMIInstances.Count > 0 Then
66:                For Each objWMIInstance In objWMIInstances
67:
68:                    If boolViewProxy Then
69:                        Set objWMIPropertySet = objWMIInstance.Properties_
70:
71:                        For Each objWMIProperty In objWMIPropertySet
72:                            DisplayFormattedProperty objWMIInstance, _
73:                                " " & objWMIProperty.Name, _
74:                                objWMIProperty.Name, _
75:                                Null
76:
77:                    Next
78:
79:                End If
80:
81:
82:                If boolSetProxy Then
83:                    intRC = objWMIInstance.SetProxySetting (strProxyAddress, intProxyPort)
84:                    If Err.Number Then ErrorHandler (Err)
85:
86:
87:                    If intRC = 0 Then
88:                        WScript.Echo "Proxy settings successfully updated."
89:                    Else
90:                        WScript.Echo "Failed to update Proxy settings (" & intRC & ")."
91:                    End If
92:
93:                End If
94:
95:            Next
96:        End If
97:
98:
99:        ]]>
100:       </script>
101:   </job>
102: </package>
```

Once command-line parsing completes (lines 62 through 100) and the WMI connection is executed (lines 102 through 106), the script retrieves all instances of the *Win32_Proxy* class (line 109). Since this class is not a singleton class, and because it uses the proxy settings as Key properties, the easiest way to retrieve the information is to start an enumeration of the available instances.

When the */ViewProxy+* switch is used, the script shows the current Windows Proxy settings (lines 114 through 124). This portion of code enumerates and displays all properties available from the *Win32_Proxy* class (lines 115 through 121). Once executed, the output will be as follows:

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

ProxyPortNumber: ..... 8080
ProxyServer: ..... http://proxy.LissWare.Net
*ServerName: ..... net-dpen6400a.LissWare.Net
```

When the */ProxyAddress* switch is supplied, the script invokes the *SetProxySetting* method of the *Win32_Proxy* class to configure the Windows Proxy LAN settings (line 127). This script is not really complicated; it simply takes advantage of the *Win32_Proxy* class capabilities.

3.3.8 Windows Product Activation provider

These providers manage the information related to the *Windows Product Activation* (WPA). The providers are implemented as instance and method providers (Table 3.17). Available for Windows XP and Windows Server 2003, the WPA is not supported on 64-bit Windows and previous Windows platforms.

Table 3.17 The Windows Product Activation Providers 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
Performance Monitoring Provider																
Win32_WIN32_COMPUTERSYSTEMWINDOWSPRODUCTACTIVATIONSETTING_Prov	Root/CIMV2	X	X				X	X	X	X	X					
Win32_WIN32_WINDOWSPRODUCTACTIVATION_Prov	Root/CIMV2	X	X				X	X	X	X	X					

Two classes are supported by the WMI providers and expose methods to perform Windows product activation. Table 3.18 lists the class names available, while Table 3.19 shows the methods exposed by the *Win32_WindowsProductActivation* class.

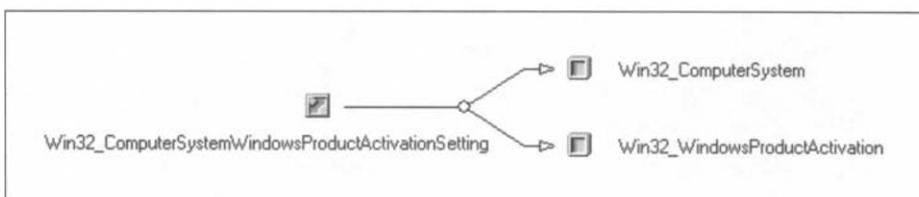
→ **Table 3.18** *The Windows Product Activation Providers Classes*

Name	Type	Comments
Win32_WindowsProductActivation	Dynamic	The Win32_WindowsProductActivation class contains properties and methods that relate to Windows Product Activation, such as: activation state, grace period, and provides the ability to activate the machine online and offline.
Win32_ComputerSystemWindowsProductActivationSetting	Association	This class represents an association between Win32_ComputerSystem and Win32_WindowsProductActivation

→ **Table 3.19** *The Windows Product Activation Providers Methods*

Name	Comments
ActivateOffline	This function is a scriptable equivalent to manual telephone activation. It permits offline activation using the Confirmation ID provided by the Microsoft Clearinghouse. In order to complete offline activation, the InstallationID for the machine must be retrieved by querying for the property. The method returns 0 on success and an error code otherwise.
ActivateOnline	The ActivateOnline method exchanges license-related data with the Microsoft Clearinghouse server and, if successful, completes system activation. It requires that the target machine be able to communicate through the Internet using the HTTPS protocol. If necessary, the SetProxySetting method should first be used to connect through a firewall. The method returns 0 on success and an error code otherwise.
GetInstallationID	GetInstallationID gets the InstallationID property and comprises the Product ID and Hardware ID, and is identical to the Installation ID displayed on the telephone activation page. Installation ID must be provided to the Microsoft Clearinghouse to obtain the corresponding Confirmation ID, which is required for the ActivateOffline method.
SetNotification	SetNotification is a function that enables or disables Windows Product Activation Notification reminders. The method returns 0 on success (or if activation is not pending) and an error code otherwise. Notification reminders are enabled by default. Note that this method does not affect Logon or event log reminders, nor does it alter the need to activate the computer.
SetProductKey	SetProductKey permits a computer's Product Key (and therefore its ProductID) to be changed or corrected. Only Product Keys that are valid for the media type (i.e., retail, volume licensing, OEM) will be accepted. The method can only be used while ActivationRequired is 1. Product KeyProduct Key is a 25-character alphanumeric string formatted in groups of five characters separated by dashes.

Note that all methods are only usable if the Windows product is not activated. Once the activation is completed, none of the methods can be used. Because WPA relates to a single computer, the *Win32_WindowsProductActivation* class is associated with the *Win32_ComputerSystem* class, as shown in Figure 3.6.



→ **Figure 3.6** *The Win32_WindowsProductActivation class is associated with the Win32_ComputerSystem class.*

It is important to understand that WPA can be executed on line or off line. While on-line activation executes via the Internet, off-line activation is executed on the phone with a Microsoft representative. In this case, the customer must provide an installation ID. Sample 3.14 implements the script logic to perform both activation types. The command-line parameters exposed by this script are as follows:

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

Usage: WMIWPA.wsf [/ViewProxy[+|-]] [/ProxyAddress:value] [/ProxyPort:value] [/ViewWPA[+|-]]
          [/ActivateOffLine:value] [/ActivateOnLine:value] [/GetInstallationID:value]
          [/SetProductKey:value] [/SetNotification:value] [/Machine:value] [/User:value]
          [/Password:value]

Options:

ViewProxy      : View the proxy settings.
ProxyAddress   : Name of the proxy server to configure.
ProxyPort      : Port number configured on the computer for access to the proxy server specified.
ViewWPA        : View the Windows Product Activation (WPA) settings.
ActivateOffLine : Activates the system offline using the confirmation ID provided by the Microsoft
                  Clearinghouse license server.
ActivateOnLine  : Exchanges license-related data with the Microsoft Clearinghouse license server;
                  if the method succeeds, it activates the system.
GetInstallationID : Retrieves the installation ID which is required to activate a system offline.
SetProductKey  : Updates the system product key for a computer.
SetNotification : Enable the notification of Number of days remaining before activation of the
                  system is required.
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)

Examples:
```

```
WMIWPA.Wsf /ViewProxy+
WMIWPA.Wsf /ProxyAddress:proxy.LissWare.Net /ProxyPort:8080

WMIWPA.Wsf /ViewWPA+
WMIWPA.Wsf /ActivateOffLine:[ConfirmationID]
WMIWPA.Wsf /ActivateOnLine
WMIWPA.Wsf /GetInstallationID
WMIWPA.Wsf /SetProductKEY:VVVVV-WWWWW-XXXXX-YYYYY-ZZZZZ
WMIWPA.Wsf /SetNotification+
```

Because proxy settings could be required for an on-line activation, the script also uses the *Win32_Proxy* class to configure the Windows Proxy LAN settings. As we have previously seen in Sample 3.13, this portion of the code is skipped in Sample 3.14 (skipped lines 153 through 186).

Sample 3.14 *Windows Product Activation*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
40:  </runtime>
41:
42:  <script language="VBScript" src=..\Functions\DisplayFormattedPropertyFunction.vbs" />
43:  <script language="VBScript" src=..\Functions\TinyErrorHandler.vbs" />
44:
45:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
46:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
47:
48:  <script language="VBScript">
49:  <![CDATA[
.:
53:  Const cComputerName = "LocalHost"
54:  Const cWMINameSpace = "Root\CIMv2"
55:  Const cWMIProxyClass = "Win32_Proxy"
56:  Const cWMIWPAClass = "Win32_WindowsProductActivation"
.:
89:  ' -----
90:  ' Parse the command line parameters
91:  If WScript.Arguments.Named.Count = 0 Then
92:      WScript.Arguments.ShowUsage()
93:      WScript.Quit
94:  End If
.:
145: strComputerName = WScript.Arguments.Named("Machine")
146: If Len(strComputerName) = 0 Then strComputerName = cComputerName
147:
148: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
149: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
150:
151: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
152:                                         strUserID, strPassword)
.:
186:
187:  ' -----
188: Set objWMIInstances = objWMIServices.InstancesOf (cWMIWPAClass)
.:
191: If objWMIInstances.Count = 1 Then
192:     For Each objWMIInstance In objWMIInstances
193:         If boolViewWPA Then
194:             Set objWMIPropertySet = objWMIInstance.Properties_
195:                 For Each objWMIPROPERTY In objWMIPropertySet
196:                     DisplayFormattedProperty objWMIInstance, _
197:                         " " & objWMIPROPERTY.Name, _
198:                         objWMIPROPERTY.Name, _
199:                         Null
200:             Next
.:
203: End If
```

```
204:     If boolGetInstallationID Then
205:         If objWMIInstance.ActivationRequired = 1 Then
206:             intRC = objWMIInstance.GetInstallationID (strInstallationID)
...
209:         If intRC = 0 Then
210:             WScript.Echo "Installation ID is '" & strInstallationID & "'."
211:         Else
212:             WScript.Echo "Failed to get Installation ID (" & intRC & ")."
213:         End If
214:     Else
215:         WScript.Echo "Unable to get the Installation ID " &
216:                     "as the Windows activation is already completed."
217:     End If
218: End If
219: If boolActivateOffLine Then
220:     If objWMIInstance.ActivationRequired = 1 Then
221:         intRC = objWMIInstance.ActivateOffline (strConfirmationID)
...
224:     If intRC = 0 Then
225:         WScript.Echo "Windows successfully activated (off-line)."
226:     Else
227:         WScript.Echo "Failed (" & intRC & ")."
228:     End If
229: Else
230:     WScript.Echo "Unable to activate Windows " &
231:                     "as the Windows activation is already completed."
232: End If
233: End If
234: If boolActivateOnLine Then
235:     If objWMIInstance.ActivationRequired = 1 Then
236:         intRC = objWMIInstance.ActivateOnline()
...
239:     If intRC = 0 Then
240:         WScript.Echo "Windows successfully activated (on-line)."
241:     Else
242:         WScript.Echo "Failed to activate Windows (" & intRC & ")."
243:     End If
244: Else
245:     WScript.Echo "Unable to activate Windows " &
246:                     "as the Windows activation is already completed."
247: End If
248: End If
249: If boolSetProductKEY Then
250:     If objWMIInstance.ActivationRequired = 1 Then
251:         intRC = objWMIInstance.SetProductKey (strProductKey)
...
254:     If intRC = 0 Then
255:         WScript.Echo "" & strProductKey & " product key successfully set."
256:     Else
257:         WScript.Echo "Failed setting '" & strProductKey & _
258:                         "' product key (" & intRC & ")."
259:     End If
260: Else
261:     WScript.Echo "Unable to set the Product Key " &
262:                     "as the Windows activation is already completed."
263: End If
264: End If
265: If Len (boolSetNotification) Then
266:     intRC = objWMIInstance.SetNotification (intSetNotificationWPA)
...
```

```

269:         If intRC = 0 Then
270:             If boolSetNotification Then
271:                 WScript.Echo "WPA Notification enabled."
272:             Else
273:                 WScript.Echo "WPA Notification disabled."
274:             End If
275:         Else
276:             WScript.Echo "Failed to set WPA notification (" & intRC & ")."
277:         End If
278:     End If
279:     Next
280: End If
...
286: 1]>
287: </script>
288: </job>
289:</package>
```

Once the command-line parameter definitions (lines 13 through 40) and parsing (lines 89 through 146) with the WMI connection are completed (lines 148 through 152), the script executes the various operations supported by the *Win32_WindowsProductActivation* class (lines 188 through 280). First, independently of the WPA operation performed, the script retrieves all instances of the *Win32_WindowsProductActivation* class (line 188). This class is not implemented as a singleton class, but since we only have one instance of the WPA in a computer system, the script only considers the first instance available (line 191). The first supported script action displays the WPA information with the following command line:

```

1:  C:\>WMIWPA.Wsf /ViewWPA+
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  ActivationRequired: ..... 1
6:  IsNotificationOn: ..... 1
7:  ProductID: ..... 55039-986-4602466-00615
8:  RemainingEvaluationPeriod: ..... 185
9:  RemainingGracePeriod: ..... 14
10: *ServerName: ..... NET-DPEP6400
```

The script logic is shown from line 194 through 200 and does not contain any particular coding that we haven't seen before.

The next WPA-supported action is the retrieval of the *InstallationID*. When performing an off-line activation, the customer must provide an *InstallationID* to the Microsoft representative over the phone. The *InstallationID* can be retrieved with the following command line:

```

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

Installation ID is '00450052973083532694579966743296021641034484695370'.
```

The script logic is available from line 204 through 217. It is important to note that the script tests the state of the activation by verifying the value contained in the *ActivationRequired* property (line 205). This verification is made for every WPA action, since the *Win32_WindowsProductActivation* methods are only executable when the product is not activated (lines 220, 235, and 250). Next, the *GetInstallationID* method is invoked. When the execution is successful (line 209), the parameter passed during the method invocation (line 206) contains the *InstallationID* required by the Microsoft representative.

In exchange, the Microsoft representative will provide another number, which is a *ConfirmationID*, that is required to perform the off-line activation. This is the next action supported by Sample 3.14. The following command line must be used:

```
C:>WMIWPA.Wsf /ActivateOffLine:[ConfirmationID]
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Windows successfully activated (off-line).
```

The script logic is available from line 219 through 233. Although the script follows the same logic as before, it invokes the *ActiveOffLine* method, which requires the *ConfirmationID* as parameter (line 221).

The next WPA action performs the on-line Product Activation. In this case the command line to use is:

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

Windows successfully activated (on-line).
```

The script logic is available from line 234 through 248. Again, the logic is exactly the same as before. However, the *ActiveOnLine* method does not require any parameter (line 236).

While the product key is given at installation time, the *SetProductKey* method of the *Win32_WindowsProductActivation* class allows the modification of the product key before activation. The following command line must be used with a valid product key:

```
C:>WMIWPA.Wsf /SetProductKEY:[VVVV-WWWWW-XXXXX-YYYYY-ZZZZZ]
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

'VVVV-WWWWW-XXXXX-YYYYY-ZZZZZ' product key successfully set.
```

The script logic is available from line 249 through 264. The *SetProductKey* method requires the product key as a parameter (line 251).

If the product is not activated, the system sends notification of the remaining days before activation is mandatory. It is possible to enable or disable this notification with the script by using the following command line:

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

WPA Notification enabled.
```

Here again, the script logic for this action is exactly the same as the previous WPA operation (lines 265 through 278). The method to configure the notification is invoked at line 266. This method requests an integer value as parameter. A value of 0 means “disable” and a value of 1 means “enable.”

3.3.9 Windows Installer provider

The *Windows Installer* provider is not always available by default for all Windows platforms! To install this provider, you should perform the following tasks:

- **For Windows XP:** It is included in the system installation by default.
- **For Windows Server 2003:** From Control Panel, select Add/Remove Programs. Next, select Add/Remove Windows Components; then, in the Windows Components Wizard, select Management and Monitoring Tools (see Figure 3.7).

Finally, select WMI *Windows Installer* provider, and then click OK. Follow the steps in the wizard to complete the installation.

- **For Windows 2000 and Windows NT 4.0:** The Windows Installer package for Windows NT 4.0 and 2000 is available at <http://www.microsoft.com/downloads/release.asp?releaseid=32832>.

The WMI *Windows Installer* provider allows WMI-enabled scripts or applications to access information collected from Windows Installer-compliant applications. The provider mirrors the functionality of the Windows Installer. It can read any property of a software product installation. In addition, due to the WMI architecture, the *Windows Installer* provider makes available to remote users the set of procedures that the Windows Installer

Figure 3.7
Adding the
Windows Installer
provider under
Windows Server
2003.



makes available to local users. This provider is implemented as an instance and a method provider (Table 3.20).

Table 3.20 The Windows Installer Provider

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
Windows Installer Provider	MSIProv	Root/CIMV2	X X					X X X X	X X X X							

The Windows Installer supports more than 50 classes. Unfortunately, it is beyond the scope of this book to review all the classes. Instead, we will give an overview of the most important classes available with their capabilities. The classes supported by the WMI *Windows Installer* provider can be classified into six categories.

- **Actions:** This category contains the classes derived from the *CIM_Action* abstract class and represents actions performed during installation, upgrade, uninstall, or application maintenance.

- **Associations:** This category contains the association classes that represent references to other WMI *Windows Installer* provider classes.
- **Checks:** This category contains the classes derived from the *CIM_Check* abstract class and represents conditions that should be met when a software feature or element is installed.
- **Core Classes:** This category contains the most important classes, because it provides most of the software installation features.
- **External Associations:** This category contains the association classes that represent the references to Win32 classes beyond the scope of the *Windows Installer* provider classes.
- **Settings:** This category contains the classes that represent instances of settings containing additional information about installations or their components.

Table 3.21 summarizes by category the WMI classes related to the Windows Installer.

The easiest way to get an immediate benefit from the Windows Installer via WMI is to work with the *Win32_Product* class. This class represents the Windows Installer-compliant products installed in a system. This class exposes seven methods with some parameters (Table 3.22).

Samples 3.15 through 3.17 illustrate the use of the *Win32_Product* methods. The script exposes the following command-line parameters:

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

Usage: WMIMSI.wsf [/Action:value] [/SoftwareFeature[+|-]] [/PackageLocation:value] [/Options:value]
                  [/AllUsers[+|-]] [/TargetLocation:value] [/ReInstallMode:value]
                  [/PackageName:value] [/PackageVersion:value]
                  [/Machine:value] [/User:value] [/Password:value]

Options:
Action      : Windows Installer action to perform. Only [View], [Install], [Admin], [Advertise],
              [ReInstall], [UnInstall] or [Upgrade] is accepted.
SoftwareFeature : View software features associated with the Windows Installer package
                  (View action only).
PackageLocation : Defines the Windows Installer package source path
                  (Install, Admin, Advertise and Upgrade actions only).
Options      : Command line options for the installation.
AllUsers     : Specifies if the package is installed for All Users
                  (Install and Advertise actions only).
TargetLocation : Defines the Windows Installer package admin installation path target
                  (Admin action only).
ReInstallMode : Defines the package reinstallation mode (Reinstall action only).
PackageName   : Name of the package (Reinstall, Upgrade and Uninstall action only).
PackageVersion : Version of the package (Reinstall, Upgrade and Uninstall action only).
```

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)
 Examples:

```
WMIMSI.Wsf /Action:View
WMIMSI.Wsf /Action:View /SoftwareFeature+
WMIMSI.Wsf /Action:Install /PackageLocation:L:\SUPPORT\TOOLS\SUPTOOLS.MSI /AllUsers+
WMIMSI.Wsf /Action:Admin /PackageLocation:L:\SUPPORT\TOOLS\SUPTOOLS.MSI
  /TargetLocation:"E:\SUPTOOLS"
WMIMSI.Wsf /Action:Advertise /PackageLocation:L:\SUPPORT\TOOLS\SUPTOOLS.MSI /AllUsers+
WMIMSI.Wsf /Action:ReInstall /ReinstallMode:Shortcut /PackageName:"Windows Support Tools"
  /PackageVersion:"5.1.2510.0"
WMIMSI.Wsf /Action:Upgrade /PackageLocation:L:\SUPPORT\TOOLS\SUPTOOLS.MSI
  /PackageName:"Windows Support Tools" /PackageVersion:"5.1.2510.0"
WMIMSI.Wsf /Action:UnInstall /PackageName:"Windows Support Tools" /PackageVersion:"5.1.2510.0"
```

Table 3.21 The Windows Installer WMI Classes

Name	Comments
Actions	
Win32_BindImageAction	The BindImage action binds each executable that needs to be bound to the DLLs imported by it by computing the virtual address of each function that is imported from all DLLs. The computed virtual address is then saved in the importing image's Import Address Table (IAT). The action works on each file installed locally.
Win32_ClassInfoAction	The RegisterClassInfo action manages the registration of COM class information with the system. In the Advertise mode the action registers all COM classes for which the corresponding feature is enabled. Else the action registers COM classes for which the corresponding feature is currently selected to be installed.
Win32_CreateFolderAction	The CreateFolder action creates empty folders for components set to be installed locally. The removal of these folders is handled by the RemoveFolders action. When a folder is newly-created, it is registered with the appropriate component identifier.
Win32_DuplicateFileAction	The DuplicateFileAction allows the author to make one or more duplicate copies of files installed by the InstallFiles executable action, either to a directory different from the original file, or to the same directory, but with a different name.
Win32_ExtensionInfoAction	The ExtensionInfoAction manages the registration of extension-related information with the system. The action registers the extension servers for which the corresponding feature is currently selected to be uninstalled.
Win32_FontInfoAction	The RegisterFonts action registers installed fonts with the system. It maps the Font.FontTitle to the path of the font file installed. The RegisterFonts action is triggered when the Component to which the Font.File_ belongs is selected for install. This implies that fonts can be made private, shared or system by making the Components to which they belong so.
Win32_MIMEInfoAction	The RegisterMIMEInfo action registers the MIME-related registry information with the system. In the Advertise mode the action registers all MIME info for servers for which the corresponding feature is enabled. Else the action registers MIME info for servers for which the corresponding feature is currently selected to be installed.
Win32_MoveFileAction	The MoveFiles action allows the author to locate files that already exist on the user's machine, and move or copy those files to a new location.
Win32_PublishComponentAction	The PublishComponents action manages the advertisement of the components that may be faulted in by other products with the system. In the Advertise mode the action publishes the all components for which the corresponding feature is enabled. Else the action publishes components for which the corresponding feature is currently selected to be installed.
Win32_RegistryAction	The WriteRegistryValues action sets up registry information that the application desires in the system Registry. The registry information is gated by the Component class. A registry value is written to the system registry if the corresponding component has been set to be installed either locally or run from source.
Win32_RemoveFileAction	The RemoveFiles action uninstalls files previously installed by the InstallFiles action. Each of these files is 'gated' by a link to an entry in the Component class; only those files whose components are resolved to the IsAbsent Action state, or the IsSource Action state IF the component is currently installed locally, will be removed. The RemoveFiles action can also remove specific author-specified files that weren't installed by the InstallFiles action. Each of these files is 'gated' by a link to an entry in the Component class; those files whose components are resolved to any 'active' Action state (i.e. not in the 'off', or NULL, state) will be removed (if the file exists in the specified directory, of course). This implies that removal of files will be attempted when the gating component is first installed, during a reinstall, and again when the gating component is removed.
Win32_RemoveIniAction	The RemoveIniValues action deletes .INI file information that the application desires to delete from .INI files. The deletion of the information is gated by the Component class. An .INI value is deleted if the corresponding component has been set to be installed either locally or run from source.
Win32_SelfRegModuleAction	The SelfRegModules action processes all the modules in the SelfReg to register the modules, if installed.
Win32_ShortcutAction	The CreateShortcuts action manages the creation of shortcuts. In the Advertise mode, the action creates shortcuts to the key files of components of features that are enabled. Advertised shortcuts are those for which the Target property is the feature of the component and the directory of the shortcut is one of the Shell folders or below one. Advertised shortcuts are created with a Microsoft installer technology Descriptor as the target. Non-advertised shortcuts are those for which the Target column in the Shortcut class a property or the directory of the shortcut is not one of the Shell folders or below one. Advertised shortcuts are created with a Microsoft installer technology Descriptor as the target. In the non-advertise mode (normal install) the action creates shortcuts to the key files of components of features that are selected for install as well as non-advertised shortcuts whose component is selected for install.

Table 3.21 *The Windows Installer WMI Classes (continued)*

Name	Comments
Actions	
Win32_TypeLibraryAction	The RegisterTypeLibraries action registers type libraries with the system. The action works on each file referenced which is triggered to be installed.
Associations	
Win32_ActionCheck	This association relates an MSI action with any locational information it requires. This location is in the form of a file and/or directory specification.
Win32_ApplicationCommandLine	The ApplicationCommandLine association allows one to identify connection between an application and its command-line access point.
Win32_CheckCheck	This association relates a MSI Check with any locational information it requires. The location is in the form of a file and/or directory specification.
Win32_ODBCDriverSoftwareElement	Since software elements in a ready to run state cannot transition into another state, the value of the phase property is restricted to In-state for CIM_SoftwareElement objects in a ready to run state.
Win32_ProductCheck	This association relates instances of CIM_Check and Win32_Product.
Win32_ProductResource	This association relates instances of Win32_Product and Win32_MSIResource.
Win32_ProductSoftwareFeatures	The CIM_ProductSoftwareFeatures association identifies the software features for a particular product.
Win32_SettingCheck	This association relates an Installer check with any setting information it requires.
Win32_ShortcutSAP	This association relates the connection between an application access point and the corresponding shortcut.
Win32_SoftwareElementAction	This association relates an MSI software element with an action that accesses the element.
Win32_SoftwareElementCheck	This association relates an MSI element with any condition or locational information that a feature may require.
Win32_SoftwareElementResource	This association relates an MSI feature with an action used to register and/or publish the feature.
Win32_SoftwareFeatureAction	This association relates an MSI feature with an action used to register and/or publish the feature.
Win32_SoftwareFeatureCheck	This association relates an MSI feature with any condition or locational information that a feature may require.
Win32_SoftwareFeatureParent	A generic association to establish dependency relationships between objects.
Win32_SoftwareFeatureSoftwareElements	CIM_Component is a generic association used to establish 'part of' relationships between Managed System Elements. For example, the SystemComponent association defines parts of a System.
Core	
Win32_Product	Instances of this class represent products as they are installed by MSI. A product generally correlates to a single installation package.
Win32_SoftwareElement	SoftwareFeatures and SoftwareElements: A 'SoftwareFeature' is a distinct subset of a Product, consisting of one or more 'SoftwareElements'. Each SoftwareElement is defined in a Win32_SoftwareElement instance, and the association between a feature and its SoftwareFeature(s) is defined in the Win32_SoftwareFeatureSoftwareElement Association. Any component can be 'shared' between two or more SoftwareFeatures. If two or more features reference the same component, that component will be selected for installation if any of these features are selected.
Win32_SoftwareFeature	SoftwareFeatures and SoftwareElements: A 'SoftwareFeature' is a distinct subset of a Product, consisting of one or more 'SoftwareElements'. Each SoftwareElement is defined in a Win32_SoftwareElement instance, and the association between a feature and its SoftwareFeature(s) is defined in the Win32_SoftwareFeatureSoftwareElement Association. Any component can be 'shared' between two or more SoftwareFeatures. If two or more features reference the same component, that component will be selected for installation if any of these features are selected.
Checks	
Win32_Condition	The Condition class can be used to modify the selection state of any entry in the Feature class, based on a conditional expression. If Condition evaluates to True, the corresponding LevelValue in the Feature class will be set to the value specified in the Condition class's Level column. Using this mechanism, any feature can be permanently disabled (by setting the Level to 0), set to be always installed (by setting the Level to 1), or set to a different install priority (by setting Level to an intermediate value). The Level may be set based upon any conditional statement, such as a test for platform, operating system, a particular property setting, etc.
Win32_DirectorySpecification	This class represents the directory layout for the product. Each instance of the class represents a directory in both the source image and the destination image. Directory resolution is performed during the CostFinalize action and is done as follows: Root destination directories: Root directories entries are those with a null Directory_Parent value or a Directory_Parent value identical to the Directory value. The value in the Directory property is interpreted as the name of a property defining the location of the destination directory. If the property is defined, the destination directory is resolved to the property's value. If the property is undefined, the ROOTDRIVE property is used instead to resolve the path. Root source directories: The value of the DefaultDir column for root entries is interpreted as the name of a property defining the source location of this directory. This property must be defined or an error will occur. Non-root destination directories: The Directory value for a non-root directory is also interpreted as the name of a property defining the location of the destination. If the property is defined, the destination directory is resolved to the property's value. If the property is not defined, the destination directory is resolved to a sub-directory beneath the resolved destination directory for the Directory_Parent entry. The DefaultDir value defines the name of the sub-directory. Non-root source directories: The source directory for a non-root directory is resolved to a sub-directory of the resolved source directory for the Directory_Parent entry. Again, the DefaultDirvalue defines the name of the sub-directory.
Win32_EnvironmentSpecification	Instances of this class contain information about any environment variables that may need to be registered for their associated products installation.
Win32_FileSpecification	Each instance of this class represents a source file with its various attributes, ordered by a unique, nonlocalized identifier. For uncompressed files, the File property is ignored, and the FileName column is used for both the source and destination file name. You must set the 'Uncompressed' bit of the Attributes column for any file that is not compressed in a cabinet.
Win32_IniFileSpecification	This class contains the JNI information that the application needs to set in an .INI file. The JNI file information is written out when the corresponding component has been selected to be installed, either locally or run from source.
Win32_LaunchCondition	The LaunchCondition class is used by the LaunchConditions action. It contains a list of conditions, all of which must be satisfied for the action to succeed.

Table 3.21 *The Windows Installer WMI Classes (continued)*

Name	Comments
Checks	
Win32_ODBCDataSourceSpecification	This association relates an MSI check with any setting information it requires.
Win32_ODBCDriverSpecification	This class represents any ODBC drivers that are to be installed as part of a particular product.
Win32_ODBCTranslatorSpecification	Instances of this class represent any ODBC Translators that are included as part of a products installation.
Win32_ProgIDSpecification	Instances of this class represent and ProgIDs that need to be registered during a given installation.
Win32_ReserveCost	This optional class allows the author to 'reserve' a specified amount of disk space in any directory, depending on the installation state of a component. Reserving cost in this way could be useful for authors who want to ensure that a minimum amount of disk space will be available after the installation is completed. For example, this disk space might be reserved for user documents, or for application files (such as index files) that are created only after the application is launched following installation. The ReserveCost class also allows custom actions to specify an approximate cost for any files, registry entries, or other items, that the custom action might install.
Win32_ServiceSpecification	Instances of this class represent the services that are to be installed along with an associated package.
Win32_SoftwareElementCondition	Instances of this class represent conditional checks that must be evaluated to TRUE before their associated Win32_SoftwareElement can be installed.
External Associations	
Win32_InstalledSoftwareElement	The InstalledSoftwareElement association allows one to identify the Computer System a particular Software element is installed on.
Win32_ServiceSpecificationService	Represents instances of Win32_ServiceSpecification and Win32_Service.
Settings	
Win32_Binary	Instances of this class represent binary information (such as bitmaps, icons, executables, etc.) that are used by an installation.
Win32_MSIResource	Represents any resources that are used by the Installer during the course of an installation, patch, or upgrade.
Win32_ODBCAttribute	The Setting class represents configuration-related and operational parameters for one or more ManagedSystemElement(s). A ManagedSystemElement may have multiple Setting objects associated with it. The current operational values for an Element's parameters are reflected by properties in the Element itself or by properties in its associations. These properties do not have to be the same values present in the Setting object. For example, a modem may have a Setting baud rate of 56Kb/sec but be operating at 19.2Kb/sec.
Win32_ODBCSourceAttribute	The Setting class represents configuration-related and operational parameters for one or more ManagedSystemElement(s). A ManagedSystemElement may have multiple Setting objects associated with it. The current operational values for an Element's parameters are reflected by properties in the Element itself or by properties in its associations. These properties do not have to be the same values present in the Setting object. For example, a modem may have a Setting baud rate of 56Kb/sec but be operating at 19.2Kb/sec.
Win32_Patch	Instances of this class represent individual patches that are to be applied to a particular file and whose source resides at a specified location.
Win32_PatchPackage	The PatchPackage class describes all patch packages that have been applied to this product. For each patch package, the unique identifier for the patch is provided, along with information about the media image the on which the patch is located.
Win32_Property	This table contains the property names and values for all defined properties in the installation. Properties with Null values are not present in the table.
Win32_ServiceControl	Instances of this class represent instructions for controlling both installed and uninstalled services.

Table 3.22 *The Win32_Product Class Methods*

Method name	Comments
Admin	
PackageLocation	The path to the package that is to be administrated.
TargetLocation	The location for the administrative image to be installed.
Options	The command-line options for the upgrade.
Advertise	
PackageLocation	The path to the package that is to be administrated.
Options	The command-line options for the upgrade.
AllUsers	Indicates whether the operation should be applied to the current user (FALSE) or all users on the machine (TRUE).
Configure	
InstallState	Default, Local, Source
InstallLevel	Default, Minimum, Maximum
Install	
PackageLocation	The path to the package that is to be administrated.
Options	The command-line options for the upgrade.
AllUsers	Indicates whether the operation should be applied to the current user (FALSE) or all users on the machine (TRUE).
Reinstall	
ReinstallMode	Perform the package installation in the specified mode. The mode can be one of the following values: FileModeMissing (1), FileModeOlderVersion (2), FileModeEqualVersion (3), FileModeExact (4), FileModeVerify (5), FileModeReplace (6), UserData (7), MachineData(8), Shortcut (9), Package (10).
Uninstall	
Upgrade	
PackageLocation	The command-line options for the upgrade.
Options	Indicates whether the operation should be applied to the current user (FALSE) or all users on the machine (TRUE).

The first portion of the script is shown in Sample 3.15. Written in Jscript, and similar to previous scripts, this sample starts with the command-line parameter definitions (skipped lines 13 through 39) and parsing (skipped lines 92 through 246). Next, it contains the coding logic to display information about the installed Windows Installer-compliant applications (lines 263 through 359).

→ **Sample 3.15** *Managing Windows Installer packages (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
39:  </runtime>
40:
41:  <script language="VBScript" src=..\\Functions\\DecodeInstallStateFunction.vbs" />
42:
43:  <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertyFunction.vbs" />
44:  <script language="VBScript" src=..\\Functions\\TinyErrorHandler.vbs" />
45:
46:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
47:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
48:
49:  <script language="JScript">
50:  <![CDATA[
51:
52:  var cComputerName = "LocalHost";
53:  var cWMINameSpace = "Root/CIMv2";
54:  var cWMIClass = "Win32_Product";
55:
56:  var cVIEW = 1;
57:  var cADMIN = 2;
58:  var cADVERTISE = 3;
59:  var cINSTALL = 4;
60:  var cREINSTALL = 5;
61:  var cUNINSTALL = 6;
62:  var cUPGRADE = 7;
.:
92: // -----
93: // Parse the command line parameters
94: if (WScript.Arguments.Named.Count == 0)
95:  {
96:    WScript.Arguments.ShowUsage();
97:    WScript.Quit();
98:  }
.:
242: strComputerName = WScript.Arguments.Named("Machine");
243: if (strComputerName == null)
244:  {
245:    strComputerName = cComputerName;
246:  }
247:
248: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault;
249: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate;
250:
251: try
```

```
252:     {
253:         objWMIServices = objWMIConnector.ConnectServer(strComputerName, cWMINamespace,
254:                                         strUserID, strPassword);
255:     }
...
261:     switch (intAction)
262:     {
263:         // VIEW -----
264:         case cVIEW:
265:             try
266:             {
267:                 objWMIInstances = objWMIServices.InstancesOf (cWMIClass);
268:             }
...
274:             enumWMIInstances = new Enumerator (objWMIInstances);
275:             for (;! enumWMIInstances.atEnd(); enumWMIInstances.moveNext())
276:             {
277:                 objWMIInstance = enumWMIInstances.item();
278:
279:                 WScript.Echo (" - " + objWMIInstance.Caption + " " + strDashes);
280:                 objWMIPropertySet = objWMIInstance.Properties_;
281:                 enumWMIPropertySet = new Enumerator (objWMIPropertySet);
282:                 for (;! enumWMIPropertySet.atEnd(); enumWMIPropertySet.moveNext())
283:                 {
284:                     objWMIProperty = enumWMIPropertySet.item();
285:
286:                     switch (objWMIProperty.Name)
287:                     {
288:                         case "Caption":
289:                             break;
290:
291:                         case "InstallState":
292:                             DisplayFormattedProperty (objWMIInstance,
293:                                         " " + objWMIProperty.Name,
294:                                         InstallState (objWMIProperty.Value),
295:                                         null);
296:                             break;
297:                         default:
298:                             DisplayFormattedProperty (objWMIInstance,
299:                                         " " + objWMIProperty.Name,
300:                                         objWMIProperty.Name,
301:                                         null);
302:                     }
303:                 }
304:
305:                 if (boolSoftwareFeature)
306:                 {
307:                     try
308:                     {
309:                         objWMIAssocInstances = objWMIServices.ExecQuery
310:                                         ("Associators of [" +
311:                                         objWMIInstance.Path_.RelPath + ") Where " +
312:                                         "ResultClass=Win32_SoftwareFeature");
313:                     }
...
319:                     if (objWMIAssocInstances.Count != 0)
320:                     {
321:                         enumWMIAssocInstances = new Enumerator (objWMIAssocInstances);
322:                         for(;!enumWMIAssocInstances.atEnd();enumWMIAssocInstances.moveNext())
323:                         {
324:                             objWMIAssocInstance = enumWMIAssocInstances.item();
325:
326:                             WScript.Echo ("\n - Software feature for '" +
```

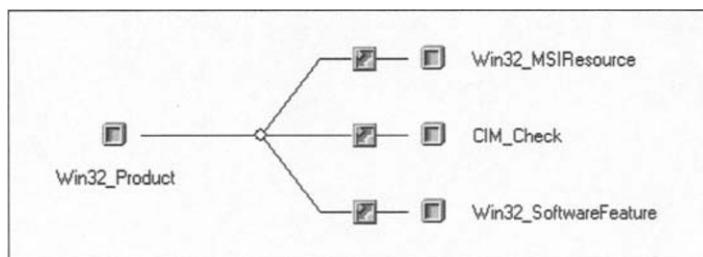
```

327:                                     objWMIInstance.Caption + " " + strDashes);
328:                                     objWMIPropertySet = objWMIAssocInstance.Properties_;
329:                                     enumWMIPropertySet = new Enumerator (objWMIPropertySet);
330:                                     for(;! enumWMIPropertySet.atEnd(); enumWMIPropertySet.moveNext())
331:                                     {
332:                                         objWMIProperty = enumWMIPropertySet.item();
333:
334:                                         switch (objWMIProperty.Name)
335:                                         {
336:                                             case "Caption":
337:                                                 break;
338:
339:                                             case "InstallState":
340:                                                 DisplayFormattedProperty (objWMIAssocInstance,
341:                                                               " " + objWMIProperty.Name,
342:                                                               InstallState (objWMIProperty.Value),
343:                                                               null);
344:                                                 break;
345:                                             default:
346:                                                 DisplayFormattedProperty (objWMIAssocInstance,
347:                                                               " " + objWMIProperty.Name,
348:                                                               objWMIProperty.Name,
349:                                                               null);
350:
351:                                         }
352:                                     }
353:                                 }
354:                             }
355:
356:                             WScript.Echo();
357:                         }
358:
359:                         break;
...:
...:
...:
```

Although Jscript is used, the logic from a WMI standpoint is exactly the same as used in many previous scripts. The script requests the collection of instances available from the *Win32_Product* class (line 267). Next, it creates an enumerator object to display properties of each *Win32_Product* instance (lines 280 through 303). If the /SoftwareFeature+ switch is given on the command line, the script displays some extra information from the *Win32_SoftwareFeature* class, which is associated with the *Win32_Product* class (Figure 3.8).

Figure 3.8

The classes associated with the *Win32_Product* class.



Once the collection of associated instances is created, the properties of each associated instance available in the collection are displayed (lines 306 through 354).

The result obtained for the Microsoft Support Tools Windows Installer package is as follows:

```
1: C:\>WMIMSI.Wsf /Action:View /SoftwareFeature+ /Machine:MyRemoteSystem.LissWare.Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Windows Support Tools -----
6: Description: ..... Windows Support Tools
7: *IdentifyingNumber: ..... {8398B542-3CC4-44D9-83DF-696CCE70124B}
8: InstallDate: ..... 20020127
9: InstallDate2: ..... 27-01-2002
10: InstallState: ..... Installed
11: *Name: ..... Windows Support Tools
12: PackageCache: ..... C:\WINDOWS\Installer\566eb2.msi
13: Vendor: ..... Microsoft Corporation
14: *Version: ..... 5.1.2510.0
15:
16: - Software feature for 'Windows Support Tools' -----
17: Accesses: ..... 0
18: Attributes: ..... 17
19: Description: ..... Minimum tools that would be installed.
20: *IdentifyingNumber: ..... {8398B542-3CC4-44D9-83DF-696CCE70124B}
21: InstallDate: ..... 27-01-2002
22: InstallState: ..... Local
23: LastUse: ..... 19800000*****.000000+****
24: *Name: ..... FeRequired
25: *ProductName: ..... Windows Support Tools
26: Vendor: ..... Microsoft Corporation
27: *Version: ..... 5.1.2510.0
28:
29: - Software feature for 'Windows Support Tools' -----
30: Accesses: ..... 0
31: Attributes: ..... 17
32: Description: ..... Windows Support Tools
33: *IdentifyingNumber: ..... {8398B542-3CC4-44D9-83DF-696CCE70124B}
34: InstallState: ..... Local
35: LastUse: ..... 19800000*****.000000+****
36: *Name: ..... FeFullKit
37: *ProductName: ..... Windows Support Tools
38: Vendor: ..... Microsoft Corporation
39: *Version: ..... 5.1.2510.0
40:
41: - Software feature for 'Windows Support Tools' -----
42: Accesses: ..... 0
43: Attributes: ..... 17
44: Description: ..... These are all optional tools.
45: *IdentifyingNumber: ..... {8398B542-3CC4-44D9-83DF-696CCE70124B}
46: InstallState: ..... Absent
47: LastUse: ..... 19800000*****.000000+****
48: *Name: ..... FeOptional
49: *ProductName: ..... Windows Support Tools
50: Vendor: ..... Microsoft Corporation
51: *Version: ..... 5.1.2510.0
```

Because WMI is DCOM based, it is possible to access the information available from a remote Windows Installer instance. This remains valid for any methods exposed by the *Win32_Product* class. The next portion of the script sample (Sample 3.16) addresses the Installation, the Admin Installation, and the Advertisement of a Windows Installer-compliant application. The use of these methods in combination with the **/Machine** switch and the **/User** and **/Password** switches allows the installation of the Windows Installer package on a remote computer from the command line. This is exactly where WMI leverages the power of the Windows Installer, since the Windows Installer is local only.

→ **Sample 3.16** *Managing Windows Installer packages (Part II)*

```
...:  
...:  
...:  
359:         break;  
360: // INSTALL -----  
361: case cINSTALL:  
362:     objWMIClass = objWMIServices.Get (cWMIClass);  
363:     WScript.Echo ("Installing ...");  
364:  
365:     intRC = objWMIClass.Install (strPackageLocation,  
366:                                     strOptions,  
367:                                     boolAllUsers);  
368:     if (intRC == 0)  
369:     {  
370:         WScript.Echo ("Package '" + strPackageLocation + " successfully installed.");  
371:     }  
372:     else  
373:     {  
374:         WScript.Echo ("Failed to install package '" + strPackageLocation +  
375:                         "' (" + intRC + ").");  
376:     }  
377:  
378:         break;  
379: // ADMIN -----  
380: case cADMIN:  
381:     objWMIClass = objWMIServices.Get (cWMIClass);  
382:     WScript.Echo ("Installing ...");  
383:  
384:     intRC = objWMIClass.Admin (strPackageLocation,  
385:                                 strTargetLocation,  
386:                                 strOptions);  
387:     if (intRC == 0)  
388:     {  
389:         WScript.Echo ("Admin installation of package '" + strPackageLocation +  
390:                         "' successfully completed.");  
391:     }  
392:     else  
393:     {  
394:         WScript.Echo ("Failed to install package '" + strPackageLocation +  
395:                         "' (" + intRC + ").");  
396:     }  
397:  
398:         break;  
399: // ADVERTISE -----
```

```

400:     case cADVERTISE:
401:         objWMIClass = objWMIServices.Get (cWMIClass);
402:         WScript.Echo ("Advertising ...");
403:
404:         intRC = objWMIClass.Advertise (strPackageLocation,
405:                                         strTargetLocation,
406:                                         boolAllUsers);
407:         if (intRC == 0)
408:             {
409:                 WScript.Echo ("Package '" + strPackageLocation +
410:                             "' successfully advertised.");
411:             }
412:         else
413:             {
414:                 WScript.Echo ("Failed to advertise package '" + strPackageLocation +
415:                             "' (" + intRC + ").");
416:             }
417:
418:         break;
...:
...:
...:

```

Because these three operations do not relate to a specific application package, the three *Win32_Product* methods (“Install,” “Admin,” and “Advertise”) are defined in the CIM repository as static methods (*Static* qualifier). This justifies the class instance creation at line 362 for the Installation, at line 381 for the Admin Installation, and at line 401 for the Advertisement. For these three operations, the script logic is exactly the same in each case. Based on the command-line parameters given, the corresponding method with its set of parameters is invoked. Note that the **/Options** command-line parameter corresponds to some optional command-line parameters specific to the Windows Installer-compliant package (see Table 3.22).

The last portion of the script is shown in Sample 3.17 and shows the script logic to use the *Reinstall*, the *Uninstall*, and the *Upgrade* methods of the *Win32_Product* class. Note that these methods are specific to a *Win32_Product* instance. This is why the script starts by retrieving the Windows Installer application that corresponds to the **/PackageName** and **/Package-Version** command-line switches (lines 423 through 426 for the “Reinstall,” lines 467 through 470 for the “Uninstall,” and lines 511 through 514 for the “Upgrade”).

Sample 3.17 *Managing Windows Installer packages (Part III)*

```

...:
...:
...:
418:         break;
419: // REINSTALL -----
420: case cREINSTALL:
421:     try
422:     {

```

```
423:     objWMIInstances = objWMIServices.ExecQuery
424:             ("Select * From " + cWMIClass +
425:             " Where Name='" + strPackageName +
426:             "' And Version='" + strPackageVersion + "'");
427:         }
428:     ...
429:     if (objWMIInstances.Count != 0)
430:     {
431:         WScript.Echo ("Found package '" + strPackageName +
432:                     "' (" + strPackageVersion + ") ... Reinstalling ...");
433:         enumWMIInstances = new Enumerator (objWMIInstances);
434:         for (; ! enumWMIInstances.atEnd(); enumWMIInstances.moveNext())
435:         {
436:             objWMIInstance = enumWMIInstances.item();
437:
438:             intRC = objWMIInstance.ReInstall (intReinstallMode);
439:             if (intRC == 0)
440:             {
441:                 WScript.Echo ("Package '" + strPackageName +
442:                             "' successfully reinstalled.");
443:             }
444:             else
445:             {
446:                 WScript.Echo ("Failed to reinstall package '" + strPackageName +
447:                             "' (" + intRC + ").");
448:             }
449:         }
450:     }
451:     else
452:     {
453:         WScript.Echo ("Package '" + strPackageName +
454:                     "' (" + strPackageVersion + ") not found.");
455:     }
456: }
457: break;
458: // UNINSTALL -----
459: case cUNINSTALL:
460:     try
461:     {
462:         objWMIInstances = objWMIServices.ExecQuery
463:             ("Select * From " + cWMIClass +
464:             " Where Name='" + strPackageName +
465:             "' And Version='" + strPackageVersion + "'");
466:     }
467:     ...
468:     if (objWMIInstances.Count != 0)
469:     {
470:         WScript.Echo ("Found package '" + strPackageName +
471:                     "' (" + strPackageVersion + ") ... Uninstalling ...");
472:
473:         enumWMIInstances = new Enumerator (objWMIInstances);
474:         for (; ! enumWMIInstances.atEnd(); enumWMIInstances.moveNext())
475:         {
476:             objWMIInstance = enumWMIInstances.item();
477:
478:             intRC = objWMIInstance.UnInstall();
479:             if (intRC == 0)
480:             {
481:                 WScript.Echo ("Package '" + strPackageName +
482:                             "' successfully uninstalled.");
483:             }
484:         }
485:     }
486: }
```

```

494:             {
495:                 WScript.Echo ("Failed to uninstall package '" + strPackageName +
496:                             "' (" + intRC + ").");
497:             }
498:         }
499:     }
500: else
501: {
502:     WScript.Echo ("Package '" + strPackageName +
503:                     "' (" + strPackageVersion + ") not found.");
504: }
505:
506: break;
507: // UPGRADE -----
508: case cUPGRADE:
509:     try
510:     {
511:         objWMIInstances = objWMIServices.ExecQuery
512:                         ("Select * From " + cWMIClass +
513:                          " Where Name='" + strPackageName +
514:                          "' And Version=''" + strPackageVersion + "'");
515:     }
516: ...
517: if (objWMIInstances.Count != 0)
518: {
519:     WScript.Echo ("Found package '" + strPackageName +
520:                     "' (" + strPackageVersion + ") ... Upgrading ...");
521:     enumWMIInstances = new Enumerator (objWMIInstances);
522:     for (;! enumWMIInstances.atEnd(); enumWMIInstances.moveNext())
523:     {
524:         objWMIInstance = enumWMIInstances.item();
525:         intRC = objWMIInstance.Upgrade (strPackageLocation,
526:                                         strOptions);
527:         if (intRC == 0)
528:             {
529:                 WScript.Echo ("Package '" + strPackageLocation +
530:                               "' successfully upgraded.");
531:             }
532:         else
533:             {
534:                 WScript.Echo ("Failed to upgrade package '" + strPackageName +
535:                               "' (" + intRC + ").");
536:             }
537:     }
538: }
539: else
540: {
541:     WScript.Echo ("Package '" + strPackageName +
542:                     "' (" + strPackageVersion + ") not found.");
543: }
544: }
545: else
546: {
547:     WScript.Echo ("Package '" + strPackageName +
548:                     "' (" + strPackageVersion + ") not found.");
549: }
550: }
551: ...
552: ]]>
553: </script>
554: </job>
555:</package>

```

Whether the requested Windows Installer action is to reinstall, uninstall, or upgrade an existing package, the script always uses the same technique. It

is important to note that the *Win32_Product* key uses three properties as a key. The following output shows the three keys (lines 7, 11, and 14):

```
1: C:\>WMIMSI.Wsf /Action:View /SoftwareFeature+ /Machine:MyRemoteSystem.LissWare.Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Windows Support Tools -----
6: Description: ..... Windows Support Tools
7: *IdentifyingNumber: ..... {8398B542-3CC4-44D9-83DF-696CCE70124B}
8: InstallDate: ..... 20020127
9: InstallDate2: ..... 27-01-2002
10: InstallState: ..... Installed
11: *Name: ..... Windows Support Tools
12: PackageCache: ..... C:\WINDOWS\Installer\566eb2.msi
13: Vendor: ..... Microsoft Corporation
14: *Version: ..... 5.1.2510.0
```

Although it is possible to supply the three keys on the command line to locate the corresponding *Win32_Product* instance, you will note that the *IdentifyingNumber* property key is not really easy to type from a command line (GUID number). For the script user facility, the code requests only the *Name* and the *Version* property key of the package. By using a WQL query, the script locates the corresponding Windows Installer application (lines 423 through 426 for an install, lines 467 through 470 for an uninstall, and lines 511 through 514 for an upgrade). For these three operations, the script logic is exactly the same in each case. Based on the supplied command-line parameters, the corresponding method with its set of parameters is invoked.

Since WMI is event driven, it is, of course, possible to monitor a package installation with the use of an appropriate WQL event query. Since this provider is not implemented as an event provider, the WQL event query requires the use of the **WITHIN** statement. The principle to detect the installation of a software product is quite easy, since it consists of tracking down the creation of the *Win32_Product* instances. For example, the WQL query to use would be as follows:

```
Select * From __InstanceCreationEvent Within 10 Where TargetInstance ISA 'Win32_Product'
```

3.3.10 Resultant Set of Policies (RSOP) providers

Have you ever dreamed of how good it would be to know if a specific Group Policy Object (GPO) is applied in the user or computer environment? Well, with the WMI implementation of the *Resultant Set of Policies* (RSOP), this is possible. The RSOP is the set of GPOs effectively applied to a particular user, once logged on, or computer, once booted. The rules

that determine if a particular GPO will be applied to a specific user or computer environment are complex. Briefly, the rules are determined by:

- The GPO container location in the Active Directory hierarchy (Local, Site, Domain, or Organizational Unit)
- The rights granted on the GPO for the object being considered for application of the considered GPO
- The WMI filter that applies for the GPO in question
- The order of the GPO at the container level, since one container may contain several GPO definitions or links (the one at the bottom of the list is the one with the highest priority)

As a consequence, the end result of the GPO application can vary widely. To discover the GPO that is applied or could be applied (simulation) to a particular object, the Windows XP and Windows Server 2003 Operating Systems contain two WMI providers that calculate the RSOP (see Table 3.23). Next, two additional providers expose complementary information about the GPO itself. They are not directly related to RSOP providers, but we will see that they can be useful in an RSOP context as well.

Table 3.23 *The RSOP Providers*

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
Resultant Set Of Policies Providers																
Rsop Planning Mode Provider	Root\RSOP		X									X	X			
Rsop Logging Mode Provider	Root\RSOP		X									X	X			
PolicSOM provider	Root\Policy					X	X			X	X					
PolicStatus provider	Root\Policy					X				X	X					

The two RSOP providers are located in the Root\RSOP namespace and are implemented as method providers. The two providers enabling complementary information about the GPO are located in the Root\Policy namespace. We will come back to these two providers later while examining the next script sample.

Before examining any methods or classes available, it is interesting to note that under Windows Server 2003, WMI is utilized at two different levels for the GPO.

- First, a WQL query can be used to filter the GPO appliance. Under Windows 2000, the only way to filter GPO is to create specific rights on the GPO Active Directory object. Based on the granted rights or the group memberships, the GPO is applied to a considered object. Although this way of filtering GPO is still available in Windows Server 2003, it is also possible to filter the GPO by creating a WQL query, called a WMI filter. WMI filters are nothing other than WQL data queries used to verify if there are items in the system matching the specified query. If the query result is positive, the GPO will be applied on the computer or user in question. Obviously, the knowledge of WQL is quite useful in creating these WMI filters. Figure 3.9 shows a WMI filter example for the Default Domain Controller Policy.

In Figure 3.9, we see that the WQL data query searches for systems that have the Windows SNMP service installed. With such a

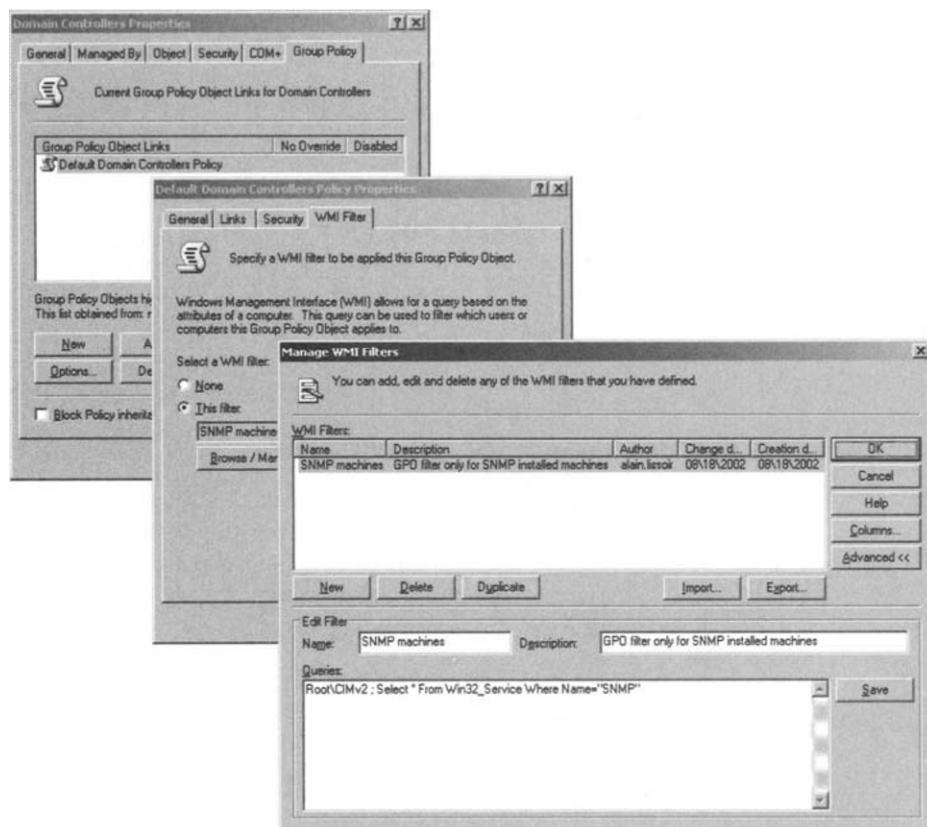


Figure 3.9 A WMI filter created for the Default Domain Controller Policy.

query, only the systems that have the SNMP service installed will have the Default Domain Controller Policy applied. You will note the particular syntax, which combines the WMI namespace and WQL data query separated by a semicolon. This syntax is not WQL specific but is just the way the information is stored in the GPO infrastructure.

- Second, WMI can be used to calculate the RSOP. Out of the box, Windows Server 2003 and Windows XP offer an MMC snap-in to analyze the RSOP. This MMC snap-in is called the “Resultant Set of Policies” snap-in. Basically this snap-in uses the two WMI providers presented in Table 3.23. In parallel to this MMC snap-in included in Windows Server 2003, Microsoft is currently developing another MMC snap-in to centrally manage the Group Policies. This MMC is called *Group Policies Management Console* (GPMC) and offers interesting functionalities to manage policies from one single interface despite the fact that GPOs are coming from different Sites, Domains or Forests. GPMC also makes use of the *RSOP* WMI providers to calculate the Resultant Set Of Policies. Beyond these functionalities, GPMC also implements an object model to manage the GPO from scripts. However, this scriptable object model has no relationship with WMI. At writing time, GPMC is still under development, therefore, more information is available through the Microsoft beta program (although GPMC is supposed to be widely available when Windows Server 2003 is released). With the help of these *RSOP* providers, and based on the GPO structure, we will see how we can develop a script to retrieve the GPO information and how to determine if some specific GPOs are applied in a specific environment.

It is important to understand that the *RSOP* providers can be used in two modes, as follows:

- **The Planning mode:** Supported by the *RSOP Planning mode* provider, this mode simulates a snapshot of the policy data that could be present on a computer or user environment by using data from Active Directory. This provider supports one single class called the *RsopPlanningModeProvider* class. This class consists of two methods, as shown in Table 3.24. The *RsopCreateSession* method creates a simulated snapshot of the policy data from a particular container (Site, Domain, or Organizational Unit) for a particular user or computer. For the simulation, it is also possible to specify a group membership and a WMI filter for the considered user or computer. The RSOP MMC snap-in, mentioned before, comes with a wizard to collect the various

parameters. Behind the scene, the snap-in uses the *RsopCreateSession* method to produce a Planning mode RSOP result. Of course, we can develop a script to produce an RSOP simulated result, but the script will simply duplicate the functionality provided by the MMC snap-in. Although technically feasible, from a management perspective, writing such a script will not provide any added value compared with the MMC snap-in.

Table 3.24 The RSOP Classes and Their Methods

Name	Type	Comments
RsopPlanningModeProvider		
RsopCreateSession	Static Method	Takes a snapshot of actual policy data for display by the RSOP UI.
RsopDeleteSession	Static Method	Deletes the snapshot of policy data made by RsopCreateSession when the data is no longer required by the RSOP UI.
RsopEnumerateUsers	Static Method	Returns the list of users whose policy data is available in logging mode.
RsopLoggingModeProvider		
RsopCreateSession	Static Method	Generates planning mode policy data.
RsopDeleteSession	Static Method	Deletes planning mode policy data.

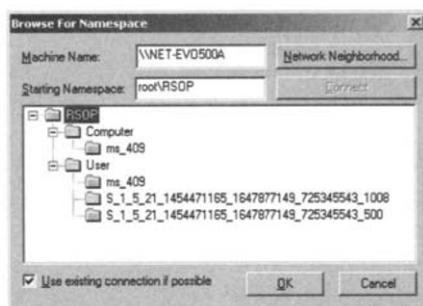
- The **Logging mode**: Supported by the *RSOP Logging mode* provider, this mode takes a snapshot of the policy data that is present on a computer or user environment. This provider supports a single class called the *RsopLoggingModeProvider* class. This class consists of three methods, as shown in Table 3.24. The *RsopCreateSession* method takes a snapshot of the policy data, which allows any application to display the RSOP data before the system overwrites or deletes it during a refresh of the policy. This mode is the most interesting one from a scripting perspective.

To understand how to work with the RSOP data, it is important to know how it is stored. We saw that the two classes for the Logging and Planning mode are available from the `Root\RSOP` namespace. To examine Logging mode RSOP data in a Windows XP or Windows Server 2003 computer, we must look at another specific namespace. Here we must make a clear distinction between two types of Logging mode RSOP data, as follows:

- The RSOP data as is once the user is logged on or when the computer is booted. In such a case, the RSOP data is available from some pre-created WMI namespaces in the CIM repository. The first namespace available is the `Root\RSOP\Computer` namespace, which exposes information about the GPOs applied on the computer. Other namespaces available include the `Root\RSOP\User\<SID>` namespace, which exposes information about the GPOs applied to users who logged on to the computer system. As you can see, the namespace uses the SID of the user in its name! The `Root\RSOP\User\<SID>`

namespace is created only during the user profile creation. If the namespace is deleted, it will not be recreated until the user profile is recreated, which means that the user profile must be deleted to recreate this namespace. However, once the namespace is created, the namespace is accordingly updated at logon time to reflect the GPOs applied on the user profile. Note that when the user profile is deleted, the RSOP namespace corresponding to the deleted user profile is also deleted. Figure 3.10 shows an example of the namespaces available.

Figure 3.10
The RSOP
subnamespaces.

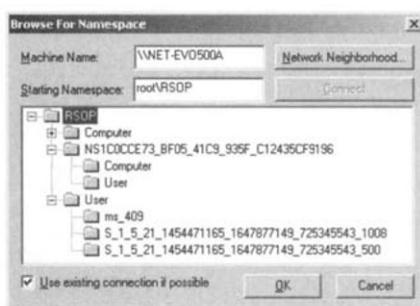


- The RSOP data that will be applied during the next policy refresh, which does not necessarily represent the applied GPO after boot or logon. The resulting *RSOP* data, as it will look after the next refresh, can be created with the *RsopCreateSession* method. The RSOP information will be stored in a dedicated namespace especially created for that purpose. To create the RSOP data for a user, it is important that the selected user has already logged on to the system. This implies that a namespace, `Root\RSOP\User\<SID>`, already exists. In such a case, the *CreateSession* method of the *RsopLoggingModeProvider* class can be invoked. Basically, when you execute the MMC *RSOP* snap-in in Logging mode, the snap-in invokes the *CreateSession* method of the *RsopLoggingModeProvider* class. The requested information will be available from this dedicated namespace. Figure 3.11 shows an example of the namespaces available.

Compared with Figure 3.10, a new namespace, called `NS1C0CCE73_BF05_41C9_935F_C12435CF9196`, is created. The subnamespaces of this namespace contain the requested information for each environment (Computer and User).

In both cases, these namespaces contain a collection of classes that represents the set of GPOs that are applied or will be applied during the next refresh cycle. Requesting instances of these classes show if a particular GPO

Figure 3.11
*The RSOP
 subnamespaces
 created during a
 CreateSession
 method invocation.*



is, or will be, effectively applied in the specified environment. The RSOP classes available are summarized in Table 3.25.

Table 3.25 *The RSOP WMI Classes*

Name	Comments
Group Policy Core Classes	
RSOP_ExtensionEventSource	Dynamic Represents client-side extensions' event log message sources.
RSOP_ExtensionEventSourceLink	Dynamic Represents the association between a client-side extension's event log message source and the extension's status.
RSOP_ExtensionStatus	Dynamic Provides information about the overall status a client-side extension's processing of policy.
RSOP_GPO	Dynamic Provides information about a GPO.
RSOP_Session	Dynamic Represents an RSOP session. There can be only one instance of this class per namespace.
RSOP_SOM	Dynamic Represents a scope of management (SOM) which can be a site, domain, organizational unit, or local scope.
RSOP_GPLink	Association Represents the links from a site, domain, organizational unit, or local scope, to one or more GPOs.
Policy Setting Template Classes	
RSOP_PolicySetting	Dynamic The class from which client-side extensions' policy objects are inherited. An instance of this class corresponds to a specific policy setting.
RSOP_PolicySettingStatus	Dynamic Provides a link to an event in the event log that corresponds to an error that occurred while applying a specific policy setting.
RSOP_PolicySettingLink	Association Represents the association between a policy setting and the setting's status.
Registry Policy Classes	
RSOP_AdministrativeTemplateFile	Dynamic Represents an administrative template (.adm) file.
RSOP_RegistryPolicySetting	Dynamic Represents the policy object for registry or administrative template extension.
Application Management Policy Classes	
RSOP_ApplicationManagementCategory	Dynamic Represents the list of programs in the Add or Remove Programs Control Panel utility.
RSOP_ApplicationManagementPolicySetting	Dynamic Represents the policy data for application management extension.
Folder Redirection and Script Policy Classes	
RSOP_FolderRedirectionPolicySetting	Dynamic Provides information about a folder-redirection extension.
RSOP_ScriptCmd	Dynamic Represents a script command and its parameters.
RSOP_ScriptPolicySetting	Dynamic Represents a script setting.
Security Policy Classes (Only available in the Computer RSOP namespace)	
RSOP_AuditPolicy	Dynamic Represents the security setting for a local group policy that relates to the auditing of an event type. Events can include, among others, system events and account management events.
RSOP_File	Dynamic Represents a security policy setting that defines the access permissions and audit settings for a securable file system object.
RSOP_RegistryKey	Dynamic Represents a security policy setting that defines the access permissions and audit settings for a particular registry key.
RSOP_RegistryValue	Dynamic Represents specific security-related registry values.
RSOP_RestrictedGroup	Dynamic Represents a security policy setting that defines the members of a restricted (security-sensitive) group.
RSOP_SecurityEventLogSettingBoolean	Dynamic Represents a security policy setting that determines whether or not guests can access the system, application and security event logs.
RSOP_SecurityEventLogSettingNumeric	Dynamic Represents a security policy setting that determines numeric properties related to the system, application and security event logs. Properties include the number of days to retain entries and maximum log size.

Table 3.25 The RSOP WMI Classes (continued)

Name		Comments
Security Policy Classes (Only available in the Computer RSOP namespace)		
RSOP_SecuritySettingBoolean	Dynamic	Represents the Boolean security setting for an account policy. Account policies include password policies and account lockout policies.
RSOP_SecuritySettingNumeric	Dynamic	Represents the numeric security setting for an account policy. Account policies include password policies, account lockout policies, and Kerberos-related policies.
RSOP_SecuritySettings	Dynamic	Abstract class from which other RSOP security classes derive. Instances of this class are not logged. RSOP_SecuritySettings derives from the RSOP_PolicySetting class.
RSOP_SecuritySettingString	Dynamic	Represents the string security setting for an account policy.
RSOP_SystemService	Dynamic	Represents the security policy setting that defines the start-up mode and access permissions for a particular system service.
RSOP_UserPrivilegeRight	Dynamic	Represents the security setting for a local group policy that relates to the assignment of a particular user privilege.
IEAK Policy Classes		
RSOP_IEAdministrativeTemplateFile	Dynamic	Represents the abstraction for an administrative template (.adm) file for Microsoft Internet Explorer.
RSOP_IEWAPolicySetting	Dynamic	Represents the policy data for general settings related to management and customization of Internet Explorer.
RSOP_IAuthenticodeCertificate	Dynamic	Represents the details of customized settings for Internet Explorer that designate software publishers and credentials agencies as trustworthy.
RSOP_IConnectionDialUpCredentials	Dynamic	Represents the settings used by the RasDial function when establishing a dial-up (remote access) connection to the Internet using Internet Explorer.
RSOP_IConnectionDialUpSettings	Dynamic	Contains the details of a phone-book entry for connecting to the Internet; corresponds to the RASENTRY structure.
RSOP_IConnectionSettings	Dynamic	Represents the details of an Internet connection made using Internet Explorer, including details related to auto-configuration.
RSOP_IConnectionWinINetSettings	Dynamic	Represents the settings used by the RasDial function to establish a remote access connection to the Internet using the Microsoft Win32® Internet (WinInet) application programming interface (API).
RSOP_IFavoriteItem	Dynamic	Represents an item or folder in a user's Internet Explorer Favorites list.
RSOP_IFavoriteOrLinkItem	Dynamic	Parent class from which Internet Explorer Favorites, Favorite folders, and Link toolbar items (Links) are inherited.
RSOP_IELinkItem	Dynamic	Represents an Internet Explorer Links bar item (a Link).
RSOP_IPrivacySettings	Dynamic	Represents the privacy settings imported for the Internet security zone.
RSOP_IProgramSettings	Dynamic	Contains details about the imported programs to use for Internet Explorer.
RSOP_IProxySettings	Dynamic	Represents the details of a proxy server connection for Internet Explorer.
RSOP_IERegistryPolicySetting	Dynamic	Represents the abstraction for registry extension policy data for Internet Explorer.
RSOP_IESecurityContentRatings	Dynamic	Represents customized settings or attributes related to security content ratings that should be used with Internet Explorer.
RSOP_IESTatusZoneSettings	Dynamic	Represents customized settings or attributes to use with Internet Explorer for a particular security zone.
RSOP_IEToolBarButton	Dynamic	Represents the toolbar button object for Internet Explorer.
RSOP_IConnectionDialUpCredentialsLink	Association	Represents the association between an Internet Explorer Administration Kit (IEAK) policy setting and the dial-up credentials for a given Internet Explorer Internet connection.
RSOP_IConnectionDialUpSettingsLink	Association	Represents the association between an IEAK policy setting and its imported dial-up settings for a specific connection of Internet Explorer to the Internet.
RSOP_IConnectionSettingsLink	Association	Represents the association between an IEAK policy setting and the policy's Internet connection settings.
RSOP_IConnectionWinINetSettingsLink	Association	Represents the association between an IEAK policy setting and WinInet connection settings for a remote access connection to the Internet.
RSOP_IFavoriteItemLink	Association	Represents the association between an IEAK policy setting and an item or folder in a user's Internet Explorer Favorites list.
RSOP_IImportedProgramSettings	Association	Represents the association between an IEAK policy setting and its imported program settings for Internet Explorer.
RSOP_ILinkItemLink	Association	Represents the association between an IEAK policy setting and an item in a user's Internet Explorer Links bar.
RSOP_IEToolBarButtonLink	Association	Represents the association between an IEAK policy setting and a custom Internet Explorer toolbar button.
Access Method Classes		
RsopLoggingModeProvider	Dynamic	Contains methods that the RSOP UI calls to take a snapshot of actual policy data in the logging mode.
RsopPlanningModeProvider	Dynamic	Contains methods that the RSOP UI calls to create resultant policy data in a what-if (planning mode) scenario.

Note that the Security Policy classes are only available from the `Root\RSOP\Computer` namespace. The easiest way to verify whether a particular GPO is applied to a User or a Computer is to check if we have instances of the `RSOP_PolicySetting` class or one of its child classes (see Figure 3.12). This is due to the fact that an instance of the `RSOP_PolicySetting` class corresponds to a specific GPO setting.

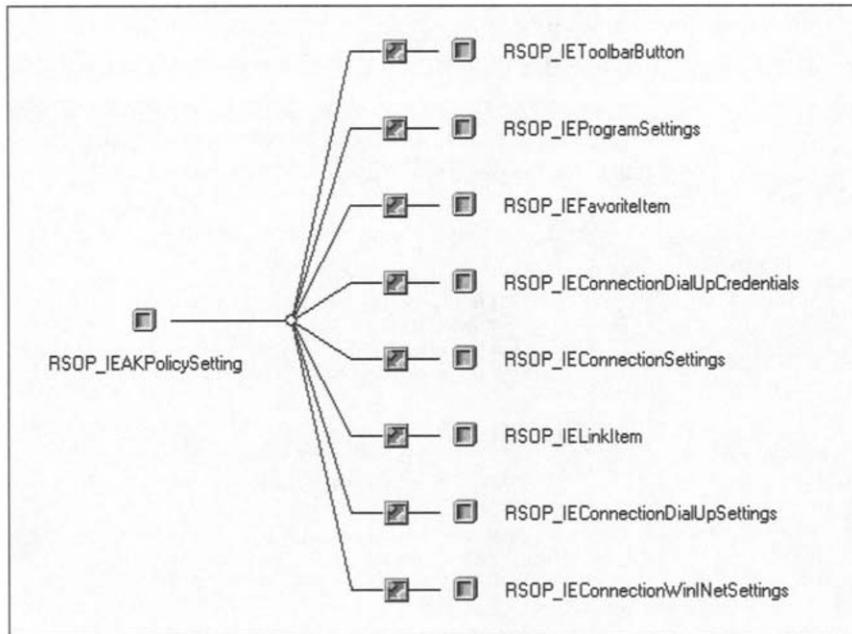
Figure 3.12
The `RSOP_PolicySetting` superclass.



Since some of these classes are associated with other classes that represent the specific settings of the GPO, by using the WMI associations in place it is possible to retrieve any type of information related to the applied GPO. For example, the `RSOP_IEAKPolicySetting`, which represents the GPO for the general settings related to management and customization of Internet Explorer, is associated with the collection of classes shown in Figure 3.13.

Now let's see how we script with the `RSOP` providers and the RSOP classes. The following scripts (Samples 3.18 through 3.24) retrieve the RSOP instances from a chosen RSOP class. If one or more instances of the `RSOP_PolicySetting` (or one of its child classes) are available, the script will display the GPO information available from the RSOP data. If the given class does not have any instance, it means that the selected GPO class is not applied for that particular environment. The following script sample supports these command-line parameters:

Figure 3.13
The RSOP_
IEAKPolicySetting
associated classes.



```

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

Usage: WMIRSOP.wsf /GPOClass:value [/UserRSOP:value] [/UserRSOPOnly[+|-]] [/ComputerRSOPOnly[+|-]]
        [/GPOFullInfo[+|-]] [/NewRSOPSession[+|-]]
        [/Machine:value] [/User:value] [/Password:value]

Options:

GPOClass      : WMI class name of the GPO to retrieve from RSOP.
UserRSOP       : Specifies User RSOP data to retrieve.
UserRSOPOnly   : Retrieve applied User GPO only.
ComputerRSOPOnly : Retrieve applied Computer GPO only.
GPOFullInfo   : Display all GPO information.
NewRSOPSession : Retrieve GPO information from a new RSOP session in logging mode.
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)

Examples:

WMIRSOP.Wsf /GPOClass:RSOP_SystemService /UserRSOP:Alain.Lissoir
              /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOP.Wsf /GPOClass:RSOP_SystemService /UserRSOP:Alain.Lissoir /GPOFullInfo+
              /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOP.Wsf /GPOClass:RSOP_IEAKPolicySetting /UserRSOP:Alain.Lissoir /UserRSOPOnly+
              /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOP.Wsf /GPOClass:RSOP_IEAKPolicySetting /UserRSOP:Alain.Lissoir /UserRSOPOnly+
              /GPOFullInfo+ /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOP.Wsf /GPOClass:RSOP_SystemService /ComputerRSOPOnly+
              /SIDResolutionDC:MyDC.LissWare.Net
  
```

```

WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /ComputerRSOPOnly+ /GPOFullInfo+
/SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /NewRSOPSession+ /UserRSOP:Alain.Lissoir
/SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /NewRSOPSession+ /UserRSOP:Alain.Lissoir
/GPOFullInfo+ /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_IEAKPolicySetting /NewRSOPSession+ /UserRSOP:Alain.Lissoir
/UserRSOPOnly+ /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_IEAKPolicySetting /NewRSOPSession+ /UserRSOP:Alain.Lissoir
/UserRSOPOnly+ /GPOFullInfo+ /SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /NewRSOPSession+ /ComputerRSOPOnly+
/SIDResolutionDC:MyDC.LissWare.Net
WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /NewRSOPSession+ /ComputerRSOPOnly+ /GPOFullInfo+
/SIDResolutionDC:MyDC.LissWare.Net

```

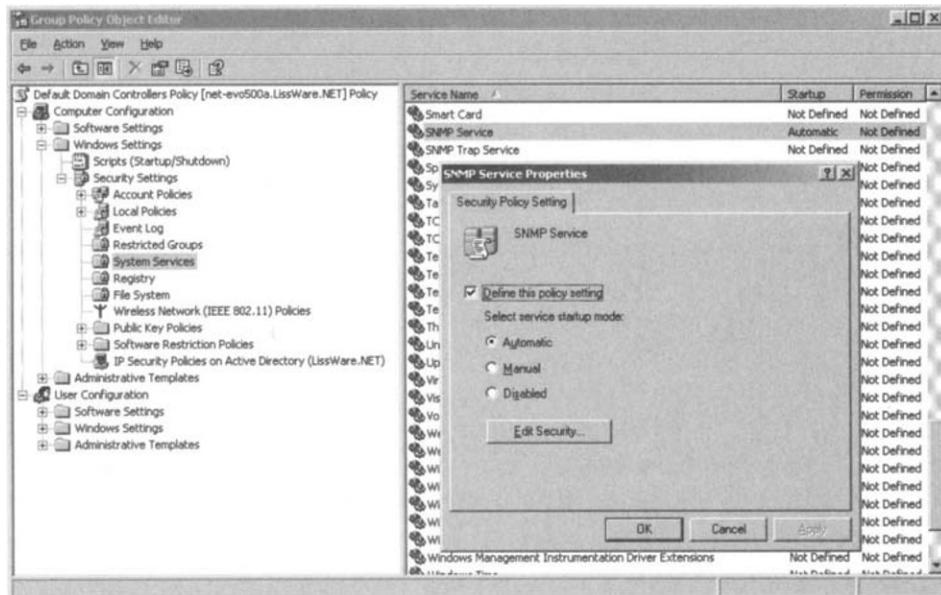


Figure 3.14 A GPO to enforce the automatic startup of the SNMP Windows service at the organizational unit level.

If a GPO is created to enforce the automatic startup of the SNMP Windows service, as shown in Figure 3.14, and if the script is started with the following command line, the obtained output would be as follows:

```

1: C:\>WMIRSOUP.Wsf /GPOClass:RSOP_SystemService /UserRSOP:Alain.Lissoir
/SIDResolutionDC:MyDC.LissWare.Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - GPO for Brussels Site (RSOP_SystemService) -----
6: creationTime: ..... 01-01-2000
7: ErrorCode: ..... 0
8: GPOID: ..... CN=(A252ACD2-2F93-44B5-ACBF-7948EC5B7080),

```

```

9:                               CN=Policies,CN=System,DC=LissWare,DC=Net
10:      id: ..... {244AD6A8-5315-43ED-9810-E6C55FA15127}
11:      name: .....
12:      *precedence: ..... 2
13:      SDDLString: ..... D:AR(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;BA)
14:                                (A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;SY)
15:                                (A;;CCLCSWLOCRRC;;;IU)S:
16:                                (AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;WD)
17:      *Service: ..... SNMP
18:      SOMID: ..... 2
19:      StartupMode: ..... 2
20:      Status: ..... 0
21:
22: - Default Domain Controllers Policy (RSOP_SystemService) -----
23:      creationTime: ..... 01-01-2000
24:      ErrorCode: ..... 0
25:      GPOID: ..... CN={6AC1786C-016F-11D2-945F-00C04FB984F9},
26:                                CN=Policies,CN=System,DC=LissWare,DC=Net
27:      id: ..... {83884976-3D07-4983-83A8-D115ADF7C777}
28:      name: .....
29:      *precedence: ..... 1
30:      SDDLString: ..... D:AR(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;BA)
31:                                (A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;SY)
32:                                (A;;CCLCSWLOCRRC;;;IU)
33:                                S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;WD)
34:      *Service: ..... SNMP
35:      SOMID: ..... 4
36:      StartupMode: ..... 2
37:      Status: ..... 1
38:
39: The 'RSOP_SystemService' GPO class does not exist in the User RSOP data.

```

Several remarks must be made about this output. First, we note that two GPOs are listed (from lines 5 and 22) instead of one. Actually, Figure 3.14 only shows the GPO settings of the “Default Domain Controllers Policy,” but, in reality, a second policy exists at the Brussels site level, as shown in Figure 3.15.

Next, we see that the GPO applied at the site level has a precedence of 2 (line 12), while the same GPO applied at the OU level has a precedence of 1. This means that the GPO at the organizational unit (OU) level (highest priority) overwrites the one at the site level (lowest priority). Keep in mind that the last applied GPO is the winner. With this in mind, it makes sense, since the GPO application order is the Local GPO first, the Site GPO next, the Domain GPO, and finally the OU GPO.

Next, we see that the *StartupMode* property (lines 19 and 36) has a value of 2, which means an automatic service startup (2=Automatic, 3=Manual, and 4=Disabled) for this particular policy. This property is defined at the level of the *RSOP_SystemService* class (look at the WMI origin definition of the property in the CIM repository).

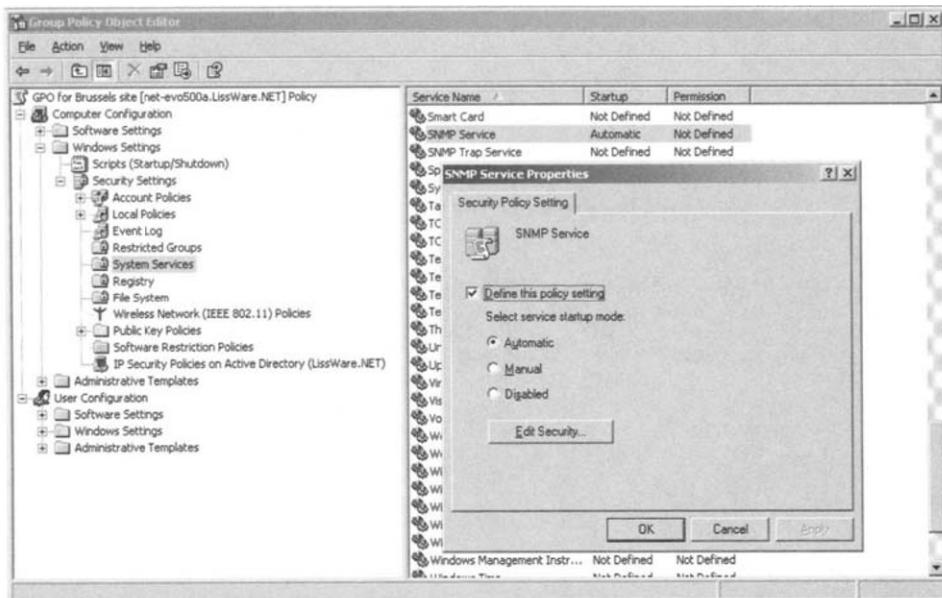


Figure 3.15 A GPO to enforce the automatic startup of the SNMP Windows service at the site level.

Based on the GPO application order, we know that only the GPO at the OU level is applied. This information is seen by looking at the *Status* property. The GPO at the site level has a status of 0 (line 20), while the GPO at the OU level has a status of 1 (line 37). By looking at the CIM origin definition of the *Status* property, we know that this property is defined at the level of the *RSOP_SecuritySettings* class. The *Status* property can take several values, as summarized in Table 3.26.

Table 3.26 The Status Property of the RSOP_SecuritySettings Class

Value	Comments
0	The system did not attempt to configure the setting.
1	The system successfully applied the policy setting for the specific item.
3	The system attempted to configure a specific policy setting but the configuration was not successful.
4	The system attempted to configure the child of a specific policy setting, but the configuration was not successful. Note that this value is valid only for configuration of file system or registry ACLs.

Based on Table 3.26, we see that the *Status* property reflects the GPO application, since the only GPO with a status of 1 (successfully applied) is the one at the OU level.

It is interesting to note that the *RSOP_SystemService* class is not available from the User environment. This makes sense, since this GPO is a Computer GPO only (line 39).

Now, if we delete the GPO SNMP Windows service setting at the OU level, the only remaining GPO is the one at the site level. Once the GPO has been refreshed, the same script execution will show the following output:

```
1: C:\>WMIRSOP.Wsf /GPOClass:RSOP_SystemService /UserRSOP:Alain.Lissoir
   /SIDResolutionDC:MyDC.LissWare.Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - GPO for Brussels Site (RSOP_SystemService) -----
6:   creationTime: ..... 01-01-2000
7:   ErrorCode: ..... 0
8:   GPOID: ..... CN=(A252ACD2-2F93-44B5-ACBF-7948EC5B7080),
9:                                CN=Policies,CN=System,DC=LissWare,DC=Net
10:  id: ..... {0F317256-2812-48AB-B3F5-B4C49943BF54}
11:  name: .....
12:  *precedence: ..... 1
13:  SDDLString: ..... D:AR(A;;CCDCLCSWRPWPDTLOCRSRCDWO;;BA)
14:                                (A;;CCDCLCSWRPWPDTLOCRSRCDWO;;SY)
15:                                (A;;CCLCSWLCRRC;;IU)
16:                                S:(AU;FA;CCDCLCSWRPWPDTLOCRSRCDWO;;WD)
17:  *Service: ..... SNMP
18:  SOMID: ..... 2
19:  StartupMode: ..... 2
20:  Status: ..... 1
21:
22: The 'RSOP_SystemService' GPO class does not exist in the User RSOP data.
```

We clearly see that the status of the GPO at the site level has changed from 0 to 1 (line 20) and that the precedence has changed from 2 to 1 (line 12).

There are many other properties available from the RSOP classes, and each of these will vary with the GPO. The purpose of this book is not to reproduce the information contained in the Microsoft SDK. Fortunately, the Microsoft SDK gives all the required information about these classes and their properties, but note that the WMI RSOP information is not available from the WMI SDK section. You must look in the "Group Policy Reference" of the "Policies and Profiles" section, in the "Setup and System Administration" general section. During the script sample discussion, we will review some of the characteristics of these classes.

Samples 3.18 through 3.24 contain the full script working with the *RSOP Logging mode* provider. As with the previous script samples, it starts with the command-line definition (skipped lines 13 through 40) and pars-

ing (lines 101 through 153). However, the WMI connection is a bit unusual, since the script must deal with several namespaces (lines 155 through 241).

Sample 3.18*Retrieving RSOP information from an applied GPO (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
40:  </runtime>
41:
42:  <script language="VBScript" src=..\\Functions\\GetSIDFromUserIDFunction.vbs" />
43:  <script language="VBScript" src=..\\Functions\\GetUserIDFromSIDFunction.vbs" />
44:  <script language="VBScript" src=..\\Functions\\ReplaceStringFunction.vbs" />
45:  <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertyFunction.vbs" />
46:  <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertiesFunction.vbs" />
47:  <script language="VBScript" src=..\\Functions\\TinyErrorHandler.vbs" />
48:
49:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
50:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
51:
52:  <script language="VBscript">
53:  <![CDATA[
.:
57:  Const cComputerName = "LocalHost"
58:  Const cWMIRSOPNameSpace = "Root\\RSOP"
59:  Const cWMIPolicyNameSpace = "Root\\Policy"
60:  Const cWMIRSOPUserNameSpace = "Root\\RSOP\\User"
61:  Const cWMIRSOPComputerNameSpace = "Root\\RSOP\\Computer"
62:  Const cWMIADNameSpace = "Root\\Directory\\LDAP"
63:  Const cWMIRSOPLoggingModeClass = "RsopLoggingModeProvider"
64:
65:  ' Diagnostic mode provider flags
66:  Const cFLAG_NO_USER = &h00000001           ' Don't get any user data
67:  Const cFLAG_NO_COMPUTER = &h00000002      ' Don't get any machine data
68:  Const cFLAG_FORCE_CREATENAMESPACE = &h00000004 ' Recreate the namespace for this snapshot.
.:
101:  ' -----
102:  ' Parse the command line parameters
103:  If WScript.Arguments.Named.Count = 0 Then
104:      WScript.Arguments.ShowUsage()
105:      WScript.Quit
106:  End If
.:
149:  strComputerName = WScript.Arguments.Named("Machine")
150:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
151:
152:  strSIDResolutionDC = WScript.Arguments.Named("SIDResolutionDC")
153:  If Len(strSIDResolutionDC) = 0 Then strSIDResolutionDC = strComputerName
154:
155:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
156:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
157:
158:  ' Connect to Root\\RSOP namespace to invoke RSOP Query available or create new one.
159:  Set objWMIROPServices = objWMILocator.ConnectServer(strComputerName, _
```

```

160:                                         cWMIRSOPNameSpace, _
161:                                         strUserID, _
162:                                         strPassword)
...
165:     ' Connect to Root\Policy namespace to gather more information about SOM
166:     Set objWMIPolicyServices = objWMIConnector.ConnectServer(strComputerName, _
167:                                                               cWMIPolicyNameSpace, _
168:                                                               strUserID, _
169:                                                               strPassword)
...
172:     Set objWMIRSOPClass = objWMIRSOPServices.Get (cWMIRSOPLoggingModeClass)
...
175:     ' Verify the correct User RSOP namespaces -----
176:     If boolComputerRSOPOnly = False Then
177:         strUserSIDRSOP = GetSIDFromUserID (strUserRSOP, _
178:                                             strSIDResolutionDC, _
179:                                             strUserID, _
180:                                             strPassword)
181:     If CheckIfRSOPLoggingModeData (objWMIRSOPClass, strUserSIDRSOP) = False Then
182:         WScript.Echo "Requested RSOP data for '" & strUserRSOP & _
183:                     "' user is not available."
184:
185:     objWMIRSOPClass.RsopEnumerateUsers arrayUserSID, intRC
...
188:     If intRC = 0 Then
189:         WScript.Echo vbCRLF & "Only the following users are " & _
190:                         "available for RSOP Logging mode:" & vbCRLF
191:         For Each strUserSID In arrayUserSID
192:             WScript.Echo "    " & GetUserIDFromSID (strUserSID, _
193:                                                 strSIDResolutionDC, _
194:                                                 strUserID, _
195:                                                 strPassword) & _
196:                     " -> " & strUserSID
197:             Next
198:         End If
199:
200:     WScript.Quit
201:     End If
202: End If
203:
...
...
...

```

The WMI connection is first established with the `Root\RSOP` namespace. This allows access to the various methods of the `RsopLoggingModeProvider` class (lines 155 through 162). Next, a second WMI connection is established with the WMI `Root\Policy` namespace (lines 165 through 169). This namespace exposes classes that contain some relevant information about the Scope of Management (SOM), which will be discussed later. Next, the script retrieves an instance of the `RsopLoggingModeProvider` class (line 172). It is interesting to note that an instance of the class must be created, since the exposed methods are static methods (*Static* qualifier).

Once the *RsopLoggingModeProvider* class instance is retrieved, the script verifies that the user name given on the command line with the /UserRSOP switch has a corresponding Root\RSOP\<SID> namespace (lines 175 through 202). To perform this verification, the script first retrieves the SID of the given user with the GetSIDFromUserID() function (lines 177 through 180). The GetSIDFromUserID() and the GetSIDFromUserID() functions were developed when examining the *Win32_UserAccount* class (see Sample 2.67, “Retrieving the SID from the UserID,” and Sample 2.68, “Retrieving the UserID from the SID”). Because these functions can establish a WMI connection to another system to perform the UserID to SID and SID to UserID resolution, the script accepts the /SIDResolutionDC switch to select the DC to use. If this switch is not given, the /Machine switch value is used, which is the accessed system for the RSOP information.

Once the SID is known, the script invokes the CheckIfRSOPLoggingModeData() function to check if the Root\RSOP\<SID> namespace exists. This function is based on the *RsopEnumerateUsers* method of the *RsopLoggingModeProvider* class (Table 3.24). We will examine this function in Sample 3.24. If the given user has no corresponding Root\RSOP\<SID> namespace, the script retrieves the list of namespaces available with the *RsopEnumerateUsers* method (line 185) and displays these namespaces with their corresponding user names (lines 189 through 197).

However, if a Root\RSOP\<SID> namespace corresponds to the given user, the script continues its execution (Sample 3.19). The next step creates a new RSOP session if the switch /NewRSOPSession+ is given on the command line.

→ **Sample 3.19** *Retrieving RSOP information from an applied GPO (Part II)*

```
...:  
...:  
...:  
203:  
204: ' Build the correct RSOP namespaces -----  
205: If boolNewRSOPSession Then  
206:   If boolUserRSOPOnly = True And boolComputerRSOPOnly = False Then  
207:     intFlags = cFLAG_FORCE_CREATENAMESPACE Or cFLAG_NO_COMPUTER  
208:   End If  
209:   If boolUserRSOPOnly = False And boolComputerRSOPOnly = True Then  
210:     intFlags = cFLAG_FORCE_CREATENAMESPACE Or cFLAG_NO_USER  
211:   End If  
212:   If boolUserRSOPOnly = False And _  
213:     boolComputerRSOPOnly = False Then  
214:     intFlags = cFLAG_FORCE_CREATENAMESPACE  
215:   End If  
216:
```

```

217: WScript.Echo "Creating new RSOP Logging mode session ..."
218:
219: objWMIRSOPClass.RsopCreateSession intFlags, _
220:                               strUserSIDRSOP, _
221:                               strTempNameSpace, _
222:                               intRC, -
223:                               intExtendedInfo
...:
226: If intRC = 0 Then
227:     WScript.Echo "New RSOP Logging mode session created." & vbCrLf
228:
229:     strWMIRSOPUserNameSpace = strTempNameSpace & "\User"
230:     strWMIRSOPComputerNameSpace = strTempNameSpace & "\Computer"
231: Else
232:     WScript.Echo "Unable to create a new RSOP Logging mode session. " & _
233:                 "(0x" & Hex(IntRC) & ")."
234:     WScript.Quit
235: End If
236: Else
237:     strWMIRSOPUserNameSpace = cWMIRSOPUserNameSpace & "\" & strUserSIDRSOP
238:     ReplaceString strWMIRSOPUserNameSpace, "-", "_"
239:
240:     strWMIRSOPComputerNameSpace = cWMIRSOPComputerNameSpace
241: End If
242:
...:
...:
...:

```

In this case, based on the presence of the `/ComputerRSOPOnly+` and the `/UserRSOPOnly+` switches, the script will determine the flags to use for the `RsopCreateSession` method invocation. Table 3.27 summarizes the various flag values. By default, if none of these switches is given on the command line, the script will request both User and Computer RSOP information.

Table 3.27 The Logging Mode and Planning Mode Flag Values

Name	Value	Description
Login mode provider flags		
FLAG_NO_USER	0x1	Don't get any user data
FLAG_NO_COMPUTER	0x2	Don't get any machine data
FLAG_FORCE_CREATENAMESPACE	0x4	Delete and recreate the namespace for this snapshot.
Planning mode provider flags		
FLAG_NO_GPO_FILTER	0x80000000	GPOs are not filtered, implies FLAG_NO_CSE_INVOKE
FLAG_NO_CSE_INVOKE	0x40000000	Only GP processing done for planning mode
FLAG_ASSUME_SLOW_LINK	0x20000000	Planning mode RSOp assumes slow link
FLAG_LOOPBACK_MERGE	0x10000000	Planning mode RSOp assumes merge loop back
FLAG_LOOPBACK_REPLACE	0x8000000	Planning mode RSOp assumes replace loop back
FLAG_ASSUME_USER_WQLFILTER_TRUE	0x4000000	Planning mode RSOp assumes all comp filters to be true
FLAG_ASSUME_COMP_WQLFILTER_TRUE	0x2000000	Planning mode RSOp assumes all user filters to be true
Error codes		
RSOP_USER_ACCESS_DENIED	0x1	User accessing the RSOp provider doesn't have access to user data.
RSOP_COMPUTER_ACCESS_DENIED	0x2	User accessing the RSOp provider doesn't have access to computer data.
RSOP_TEMPNAMESPACE_EXISTS	0x4	This user is an interactive nonadmin user, the temp snapshot namespace already exists, and the FLAG_FORCE_CREATENAMESPACE was not passed in.

When creating a new *RSOP* session, it is possible to select the *RSOP* information for only the user or for only the computer environment. The flag combination determines the *RSOP* data requested (lines 206 through 215). Because a new *RSOP* session creates a temporary namespace (see Figure 3.11), the temporary namespace name is returned in one of the method parameters (line 221). If the method invocation is successful (line 226), the temporary namespaces for the User (line 229) and the Computer *RSOP* data are initialized (line 230).

If the /NewRSOPSession+ switch is not supplied on the command line, the script initializes the namespace names in order to work with the *RSOP* data available in the Root\RSOP\Computer and Root\RSOP\User\<SID> namespaces (lines 237 through 240). Note the replacement of the dash (-), which exists in the SID by an underscore (_) (line 238). This character replacement must be done, because WMI uses the SID of the user with underscores instead of dashes to name the namespace.

Once completed, the script is ready to connect to the required *RSOP* namespaces to examine the User and/or the Computer *RSOP* data in Logging mode from the current session or from a new session, since the namespace names are initialized accordingly (Sample 3.20).

→ **Sample 3.20** *Retrieving RSOP information from an applied GPO (Part III)*

```
...:  
...:  
...:  
242:  
243: ' Get the RSOP Computer data -----  
244: If boolUserRSOPOnly = False Then  
245:   ' Connect to RSOP computer namespace  
246:   Set objWMIRSOPComputerServices = objWMIConnector.ConnectServer(strComputerName, _  
247:                                         strWMIRSOPNameSpace, _  
248:                                         strUserID, _  
249:                                         strPassword)  
...:  
252: Set objWMIRSOPInstances = objWMIRSOPComputerServices.InstancesOf (strWMIGPOClass)  
...:  
255: If objWMIRSOPInstances.Count Then  
256:   If Err.Number Then  
257:     WScript.Echo "The '" & strWMIGPOClass & "' GPO class does not " & -  
258:           "exist in the computer RSOP data." & vbCRLF  
259:   Err.Clear  
260: Else  
261:   DisplayRSOPInstances objWMIPolicyServices, _  
262:                           objWMIRSOPComputerServices, _  
263:                           objWMIRSOPInstances, _  
264:                           boolGPOFullInfo  
265: End If  
266: Else  
267:   WScript.Echo "No computer RSOP data the '" & strWMIGPOClass & _
```

```

268:           "' GPO class." & vbCRLF
269:     End If
...:
274:   End If
275:
...:
...:
...:

```

To examine the RSOP Computer information, the principle is quite simple. First, the script connects to the corresponding namespace (lines 246 through 249). Next, it requests all instances available from the RSOP class given on the command line (line 252). If there is at least one instance available (line 255), it means that a GPO exists for this class. Next, the related information is displayed (lines 261 through 264) via the DisplayRSOPInstances() function. This function will be examined in Sample 3.23.

To retrieve the RSOP User information, the principle is exactly the same as the RSOP Computer information. Only the namespace is different (Sample 3.21).

Sample 3.21 *Retrieving RSOP information from an applied GPO (Part IV)*

```

...:
...:
...:
275:
276: ' Get the RSOP User data -----
277: If boolComputerRSOPOnly = False Then
278:   ' Connect to RSOP user namespace
279:   Set objWMIRSOPUserServices = objWMILocator.ConnectServer(strComputerName, _
280:                                         strWMIRSOPUserNameSpace, _
281:                                         strUserID, _
282:                                         strPassword)
...:
285: Set objWMIRSOPInstances = objWMIRSOPUserServices.InstancesOf (strWMIGPOClass)
...:
288: If objWMIRSOPInstances.Count Then
289:   If Err.Number Then
290:     WScript.Echo "The '" & strWMIGPOClass & "' GPO class does not " & -
291:                 "exist in the User RSOP data." & vbCRLF
292:     Err.Clear
293:   Else
294:     DisplayRSOPInstances objWMIPolicyServices, _
295:                           objWMIRSOPUserServices, _
296:                           objWMIRSOPInstances, _
297:                           boolGPOFullInfo
298:   End If
299: Else
300:   WScript.Echo "No user RSOP data for the '" & strWMIGPOClass & -
301:                 "' GPO class." & vbCRLF
302: End If
303:
304: Set objWMIRSOPInstances = Nothing
305:

```

```

306:     Set objWMIRSOPUserServices = Nothing
307: End If
308:
...:
...:
...:
```

Before completion, if a new RSOP session was created, the script deletes the temporary namespace to ensure that the CIM repository is cleaned up (Sample 3.22). The *RsopDeleteSession* method of the *RsopLoggingModeProvider* class will delete the temporary namespace (line 311).

 **Sample 3.22** *Retrieving RSOP information from an applied GPO (Part V)*

```

...:
...:
...:
308:
309:     ' Delete new session RSOP namespace -----
310: If boolNewRSOPSession Then
311:     objWMIRSOPClass.RsopDeleteSession strTempNameSpace, _
312:                                     intRC
...:
315:     If intRC = 0 Then
316:         WScript.Echo "New RSOP Logging mode session deleted."
317:     Else
318:         WScript.Echo "Unable to delete the new RSOP " & _
319:                         "Logging mode session. (0x" & Hex(IntRC) & ")."
320:     End If
321: End If
322:
...:
...:
...:
```

For each RSOP data (Computer and User), the *DisplayRSOPInstances()* is invoked (Sample 3.23). To display the RSOP information, the script uses an enumeration technique (lines 345 through 462), since the RSOP instances are retrieved as a collection of instances.

 **Sample 3.23** *Retrieving RSOP information from an applied GPO (Part VI)*

```

...:
...:
...:
329:
330:     -----
331: Function DisplayRSOPInstances (objWMIPolicyServices, _
332:                                 objWMIRSOPServices, _
333:                                 objWMIRSOPInstances, _
334:                                 boolGPOFullInfo)
...:
345:     For Each objWMIRSOPInstance In objWMIRSOPInstances
346:         Set objWMIGPOInstance = objWMIRSOPServices.Get ("RSOP_GPO.id='"
347:                                         & objWMIRSOPInstance.GPOID & "')"
```

```
...:  
350:     WScript.Echo "- " & objWMIGPOInstance.Name & _  
351:             " (" & objWMIRSOPIstance.Path_.Class & ") " & _  
352:             String (60, "--")  
353:  
354:     Set objWMIPropertySet = objWMIRSOPIstance.Properties_  
355:     For Each objWMIProperty In objWMIPropertySet  
356:         DisplayFormattedProperty objWMIRSOPIstance, _  
357:             " " & objWMIProperty.Name, _  
358:             objWMIProperty.Name, _  
359:             Null  
360:     Next  
...:  
363:     Set objWMIAssocInstances = objWMIRSOPIservices.ExecQuery _  
364:                         ("Associators of {" & _  
365:                         objWMIRSOPIstance.Path_.RelPath & "}")  
366:  
367:     For Each objWMIAssocInstance In objWMIAssocInstances  
368:         WScript.Echo vbCRLF & " - " & objWMIAssocInstance.Path_.Class & _  
369:             " " & String (60, "--")  
370:         Set objWMIPropertySet = objWMIAssocInstance.Properties_  
371:         For Each objWMIProperty In objWMIPropertySet  
372:             DisplayFormattedProperty objWMIAssocInstance, _  
373:                 " " & objWMIProperty.Name, _  
374:                 objWMIProperty.Name, _  
375:                 Null  
376:         Next  
...:  
378:     Next  
...:  
382:     If boolGPOFullInfo Then  
383:         WScript.Echo vbCRLF & " - Complementary GPO information (" & _  
384:                     objWMIGPOInstance.Path_.Class & ") " & String (60, "--")  
385:  
386:     Set objWMIPropertySet = objWMIGPOInstance.Properties_  
387:     For Each objWMIProperty In objWMIPropertySet  
388:         Select Case objWMIProperty.Name  
389:             Case "securityDescriptor"  
390:  
391:             Case Else  
392:                 DisplayFormattedProperty objWMIGPOInstance, _  
393:                     " " & objWMIProperty.Name, _  
394:                     objWMIProperty.Name, _  
395:                     Null  
396:         End Select  
397:     Next  
...:  
400:     If Len (objWMIGPOInstance.filterId) Then  
401:         Set objWMIFilterInstance=objWMIPolicyServices.Get(objWMIGPOInstance.filterId)  
...:  
403:         WScript.Echo vbCRLF & " - GPO Filter (" & _  
404:                     objWMIFilterInstance.Path_.Class & ") " & String (60, "--")  
405:         Set objWMIPropertySet = objWMIFilterInstance.Properties_  
406:         For Each objWMIProperty In objWMIPropertySet  
407:             If objWMIProperty.CIMType = wbemCimtypeObject Then  
408:                 For Each varElement In objWMIProperty.Value  
409:                     DisplayFormattedProperties varElement, 6  
410:                 Next  
411:             Else  
412:                 DisplayFormattedProperty objWMIFilterInstance, _
```

```

413:                               "      & objWMIProperty.Name, _
414:                               objWMIProperty.Name, _
415:                               Null
416:               End If
417:           Next
...
419:       End If
420:
421:       Set objWMIAssocInstances = objWMIRSOPServices.ExecQuery _
422:                               ("Associators of {" & _
423:                               objWMIGPOInstance.Path_.RelPath & "}")
424:
425:       For Each objWMIAssocInstance In objWMIAssocInstances
426:           WScript.Echo vbCRLF & " - GPO Scope of Domain (" & _
427:                               objWMIAssocInstance.Path_.Class & ") " & String (60, "-")
428:           Set objWMIPropertySet = objWMIAssocInstance.Properties_
429:           For Each objWMIProperty In objWMIPropertySet
430:               DisplayFormattedProperty objWMIAssocInstance, _
431:                               "      & objWMIProperty.Name, _
432:                               objWMIProperty.Name, _
433:                               Null
434:           Next
...
436:       Next
...
440:       Set objWMIAssocInstances = objWMIRSOPServices.ExecQuery _
441:                               ("References of {" & _
442:                               objWMIGPOInstance.Path_.RelPath & "}")
443:
444:       For Each objWMIAssocInstance In objWMIAssocInstances
445:           WScript.Echo vbCRLF & " - GPO link (" & _
446:                               objWMIAssocInstance.Path_.Class & ") " & String (60, "-")
447:           Set objWMIPropertySet = objWMIAssocInstance.Properties_
448:           For Each objWMIProperty In objWMIPropertySet
449:               DisplayFormattedProperty objWMIAssocInstance, _
450:                               "      & objWMIProperty.Name, _
451:                               objWMIProperty.Name, _
452:                               Null
453:           Next
...
455:       Next
...
458:   End If
459:   WScript.Echo
...
462:   Next
...
466: End Function
467:
...
...

```

For each RSOP instance, a different level of information is retrieved:

- **Information about the RSOP instance** (lines 346 through 360): First, an instance of the *RSOP_GPO* class is retrieved. This class includes information about a GPO. To retrieve this information, the

script uses the *GPOID* property of the RSOP class (lines 346 and 347). This instance is useful to display the GPO container name (line 350). If more details about the GPO are requested by the **/GPOFullInfo+** switch, this instance will be used again. Next, the script displays the properties of the RSOP instance coming from the class given on the command line (lines 354 through 360).

- **Information about the RSOP associated instances** (lines 363 through 378): If there are some instances associated with the RSOP instance, the WQL query executed in this portion of the code (lines 363 through 365) will return a collection of instances. Next, the returned collection is enumerated (lines 367 through 378) and the information displayed (lines 370 through 376). For example, if instances of the *RSOP_IEAkPolicySetting* class are examined, it is likely that some associated instances will be available (see Figure 3.13).

Next, the script enters in a code portion, which is only executed if the **/GPOFullInfo+** switch is specified (lines 382 through 458). In such a case, the output obtained would be as follows:

```

1: C:\>WMIRSOP.Wsf /GPOClass:RSOP_SystemService /UserRSOP:Alain.Lissoir /GPOFullInfo+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - GPO for Brussels Site (RSOP_SystemService) -----
6:   creationTime: ..... 01-01-2000
7:   ErrorCode: ..... 0
8:   GPOID: ..... CN=(A252ACD2-2F93-44B5-ACBF-7948EC5B7080),
9:                                CN= Policies, CN= System, DC= LissWare, DC= Net
10:  id: ..... (6562C1CD-B998-44C4-85A8-3BB1F780E1D9)
11:  name: .....
12:  *precedence: ..... 1
13:  SDDLString: ..... D:AR(A;;CCDLCSWRPWPDTLOCRSDRCWDO;;BA)
14:                                (A;;CCDLCSWRPWPDTLOCRSDRCWDO;;SY)
15:                                (A;;CCLCSWLCRRC;;IU)
16:                                S:(AU;FA;CCDLCSWRPWPDTLOCRSDRCWDO;;WD)
17:  *Service: ..... SNMP
18:  SOMID: ..... 2
19:  StartupMode: ..... 2
20:  Status: ..... 1
21:
22: - Complementary GPO information (RSOP_GPO) -----
23:   accessDenied: ..... FALSE
24:   enabled: ..... TRUE
25:   fileSystemPath: ..... \\LissWare.Net\SysVol\
26:                                LissWare.Net\Policies\
27:                                {A252ACD2-2F93-44B5-ACBF-7948EC5B7080}\Machine
28:   filterAllowed: ..... TRUE
29:   filterId: ..... MSFT_SomFilter.ID=
30:                                "(C455EC3F-A9BF-4EA0-B224-571CA683B11C)",
31:                                Domain="LissWare.Net"
32:   guidName: ..... {A252ACD2-2F93-44B5-ACBF-7948EC5B7080}
33:   *id: ..... CN=[A252ACD2-2F93-44B5-ACBF-7948EC5B7080],
34:                                CN= Policies, CN= System, DC= LissWare, DC= Net

```

```

35:      name: ..... GPO for Brussels Site
36:      version: ..... 65537
37:
38:      - GPO Filter (MSFT_SomFilter) -----
39:          Author: ..... Alain.Lissoir
40:          ChangeDate: ..... 25-01-2002 14:46:38
41:          CreationDate: ..... 01-01-2002 14:55:53
42:          Description: ..... Only applied SNMP
43:          *Domain: ..... LissWare.Net
44:          *ID: ..... (C455EC3F-A9BF-4EA0-B224-571CA683B11C)
45:          Name: ..... Only for SNMP machines
46:
47:      - MSFT_Rule -----
48:          Query: ..... Select * From Win32_Service Where Name="SNMP"
49:          QueryLanguage: ..... WQL
50:          TargetNameSpace: ..... Root\cimv2
51:
52:      - GPO Scope of Domain (RSOP_SOM) -----
53:          blocked: ..... FALSE
54:          blocking: ..... FALSE
55:          *id: ..... CN=Brussels,CN=Sites,CN=Configuration,
56:                         DC=LissWare,DC=Net
57:          *reason: ..... 1
58:          SOMOrder: ..... 2
59:          type: ..... 2
60:
61:      - GPO link (RSOP_GPLink) -----
62:          appliedOrder: ..... 2
63:          enabled: ..... TRUE
64:          *GPO: ..... RSOP_GPO.id=
65:                         "CN={A252ACD2-2F93-44B5-ACBF-7948EC5B7080}",
66:                         CN=Policies,CN=System,DC=LissWare,DC=Net"
67:          linkOrder: ..... 2
68:          noOverride: ..... FALSE
69:          *SOM: ..... RSOP_SOM.id="CN=Brussels,CN=Sites,
70:                         CN=Configuration,DC=LissWare,DC=Net",reason=1
71:          *somOrder: ..... 1
72:
73: The 'RSOP_SystemService' GPO class does not exist in the User RSOP data.

```

The supplementary information displayed by the presence of the /GPO-FullInfo+ switch starts at line 22.

This portion of the script (Sample 3.23, lines 378 through 452) also retrieves different levels of information about the GPO:

- **Information about the RSOP_GPO instance** (lines 383 through 397): This instance is retrieved earlier in the script code (lines 346 and 347), but its related information is only displayed if the /GPOFullInfo+ switch is specified (lines 386 through 397).
- **Information about the list of rules**, expressed as WMI queries, which are evaluated on the target machine (lines 400 through 419): This portion of the code makes use of the RSOP_GPO instance previously retrieved (lines 346 and 347) and the namespace RootPolicy connection established at lines 166 through 169. It exploits the *FilterID*

property displayed in the output sample at lines 29 through 31, which contains a WMI path of a GPO filter corresponding to an instance of the *MSFT_SomFilter* class:

```
MSFT_SomFilter.ID=" {C455EC3F-A9BF-4EA0-B224-571CA683B11C} ",Domain="LissWare.Net"
```

The classes available in the *Root\Policy* namespace are listed in Table 3.28.

Table 3.28

The Root\Policy Classes

Name	Type	Comments
MSFT_SomFilterStatus	Dynamic	Represents client-side extensions' event log message sources.
MSFT_Rule	Dynamic	Represents the association between a client-side extension's event log message source and the extension's status.
MSFT_SomFilter	Dynamic	Provides information about the overall status of a client-side extension's processing of policy.

Once the instance of the WMI path is retrieved (line 401), the script displays its properties (lines 405 through 417). The output is available from line 38 through 45. It is important to note that the *Rules* property of the *MSFT_SomFilter* class contains an array of objects (embedded objects). The purpose of the syntax evaluation at line 407 is to determine if an object is contained in the property. If the test is positive, the array containing objects is enumerated (lines 408 through 410), and the properties of each object are displayed with the help of the *DisplayFormattedProperties()* function (line 409). The output is available from line 47 through 50. You will recognize the WMI filter displayed in Figure 3.9.

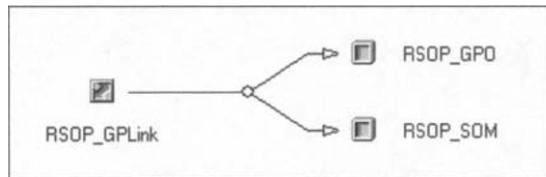
For completeness, since WMI filters are stored in Active Directory, it is possible to retrieve the filters by executing a WQL query in the *Root\directory\LDAP* namespace:

```
Select * From ads_msWMI_Som Where DS_cn=" {C455EC3F-A9BF-4EA0-B224-571CA683B11C} "
```

The WMI filters use the *ID* property of the *MSFT_SomFilter* WMI class as the CN of the Active Directory object. The Active Directory object representing a WMI filter uses an *msWMI-Som* Active Directory class. In section 3.6, we see in more depth how to work with the *Active Directory* providers.

- **Information about the Scope of Management (SOM)** (lines 421 through 436): Because we have an instance of the *RSOP_GPO* class (lines 346 and 347), it is interesting to exploit a new association not yet mentioned. This association is shown in Figure 3.16.

Figure 3.16
The RSOP_GPO association.



With this association, we retrieve instances of the *RSOP_SOM* class. This class contains information about the SOM, which could be Local, a Site, a Domain, or an OU. This information is displayed in the output example from line 52 through 59. Some relevant information is available from this class:

- The “blocked” state (line 53), which is a flag that indicates whether this SOM is blocked by a SOM lower in the hierarchy of Sites, Domains, and OUs.
- The “blocking” state (line 54), which is a flag that indicates whether this SOM blocks inheritance of policy from other SOMs higher in the hierarchy.
- The SOM (line 55) itself, which is the Brussels site in this case.
- **Information about the GPO link** (lines 440 through 455): Since we have an association in place, an association class is used with its references (lines 440 through 442). In this case, the class is the *RSOP_GPLink* class. By retrieving the instances available from this class, we can get some interesting information about the GPO (lines 61 through 71 in the output sample). For example, we can see if the GPO is enabled (line 63) and its override state (line 68).

The last piece of code of this script sample simply contains the *CheckIfRSOPLoggingModeData()* function, which was used earlier (line 181). The script code makes use of this routine to test if the *Root\RSOP\User\<SID>* namespace is available for the User name given on the command line. This function compares (line 482) the SID of the user (passed as a parameter) with the list of SIDs available from the RSOP User namespaces (line 474). If there is a match, it means that the namespace exists.

Sample 3.24 Retrieving RSOP information from an applied GPO (Part VII)

```

...:
...:
...:
467:
468: -----
469: Function CheckIfRSOPLoggingModeData (objWMIRSOPClass, strUserSIDRSOP)
...:

```

```

477:     CheckIfRSOPLoggingModeData = False
478:
479:     objWMIRSOPClass.RsopEnumerateUsers arrayUserSID, intRC
...:
482:     If intRC Then
483:         Exit Function
484:     End If
485:
486:     For Each strUserSID In arrayUserSID
487:         If strUserSID = strUserSIDRSOP Then
488:             CheckIfRSOPLoggingModeData = True
489:             Exit Function
490:         End If
491:     Next
492:
493: End Function
494:
495: ]]>
496: </script>
497: </job>
498:</package>
```

We now have an understanding of how an RSOP is represented by WMI. To monitor the appearance or the deletion of a specific GPO during a refresh cycle, WQL event queries can be submitted in the appropriate namespaces (i.e., `Root\RSOP\Computer`). If a Windows service GPO is applied to a computer, the following WQL event query should be used:

```
Select * From __InstanceCreationEvent Within 5 Where TargetInstance ISA 'RSOP_SystemService'
```

If the same GPO is removed, the following WQL event query should be used:

```
Select * From __InstanceDeletionEvent Within 5 Where TargetInstance ISA 'RSOP_SystemService'
```

Note the presence of the `WITHIN` statement, since the RSOP providers are not implemented as event providers.

If we reuse Sample 6.17 (“A generic script for asynchronous event notification”) presented in the appendix, we can easily track the GPO creation and deletion. For example, for a GPO deletion, we will have:

```

1: C:\>GenericEventAsyncConsumer.wsf "Select * From __InstanceDeletionEvent Within 5
   Where TargetInstance ISA 'RSOP_SystemService'" /NameSpace:Root\RSOP\Computer
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Sunday, 03 February, 2002 at 13:01:42: '__InstanceDeletionEvent' has been triggered.
9:     SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
10:    TargetInstance (wbemCimtypeObject)
11:        creationTime (wbemCimtypeDatetime) = (null)
12:        ErrorCode (wbemCimtypeUInt32) = 0
```

```
13:      GPOID (wbemCimtypeString) = CN={6AC1786C-016F-11D2-945F-00C04FB984F9},  
14:                                         CN=Policies,CN=System,DC=LissWare,DC=Net  
15:      id (wbemCimtypeString) = {25127460-F569-4D2C-95FB-69D47F0294B5}  
16:      name (wbemCimtypeString) =  
17:      *precedence (wbemCimtypeUInt32) = 1  
18:      SDDLString (wbemCimtypeString) = D:AR(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)  
19:                                         (A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)  
20:                                         (A;;CCLCSWLOCRRC;;IU)  
21:                                         S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)  
22:      *Service (wbemCimtypeString) = SNMP  
23:      SOMID (wbemCimtypeString) = 4  
24:      StartupMode (wbemCimtypeUInt32) = 2  
25:      Status (wbemCimtypeUInt32) = 1  
26:      TIME_CREATED (wbemCimtypeUInt64) = 03-02-2002 12:01:42 (20020203120142.177488+060)  
27:  
28: END - OnObjectReady.
```

In this sample output, we recognize the properties of the *RSOP_System-Service* class discussed in this section. Of course, as with any WQL query, it can be generalized with the use of a parent class to capture any GPO type creation and deletion. This means we have a mechanism to monitor any updates on any systems.

Since the GPO mechanism is quite complex, there is no better way than to play with the script to understand the underlying mechanisms. This is key for a good understanding of this technology.

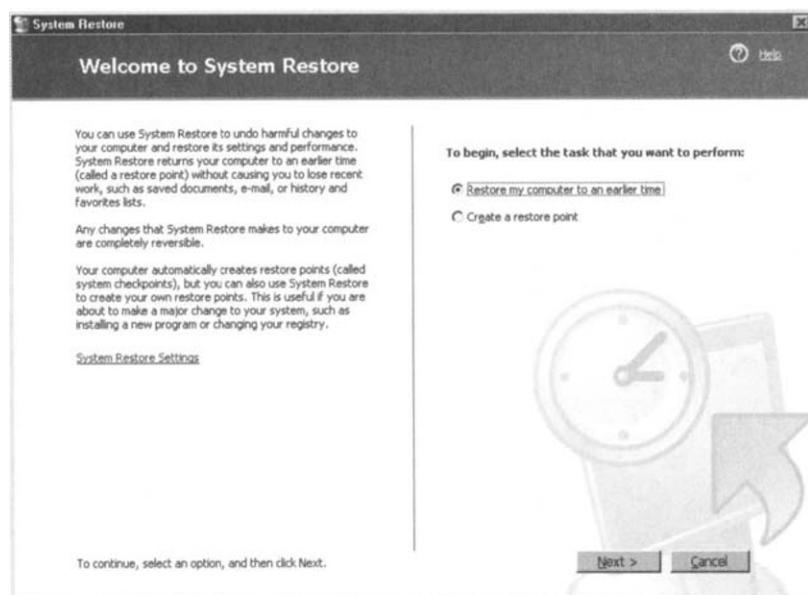
3.3.11 The System Restore provider

To undo harmful changes to a computer, Windows XP includes a feature called *System Restore*. *System Restore* returns the Operating System to an earlier situation (called a restore point) without causing the loss of all recent work. Any changes that *System Restore* makes are completely reversible. Windows XP automatically creates restore points (called system checkpoints), but it is also possible to create your own restore points. This is useful if you are about to make a major change to your system, such as installing a new program or changing your registry. The *System Restore* wizard, located in the System Tools folder of the Start Menu, allows you to perform *System Restore* operations (see Figure 3.17).

When a restore point is created, *System Restore* archives the states of a core set of system and application files with a full snapshot of the registry and some dynamic system files. To function properly, *System Restore* requires a minimum of 200 MB of free disk space on the system drive. If the free disk space falls below 50 MB on any drive, *System Restore* switches to standby mode and stops creating restore points. In such a case, all restore points are deleted. If you recover 200 MB of free disk space, the *System Restore* resumes and continues to create restore points. The set of files that is

Figure 3.17

The System Restore wizard.



monitored or excluded by *System Restore* is specified in an XML file called FILELIST.XML and located in the %windir%\System32\Restore folder. When a user is not actively using the Windows XP system, *System Restore* compresses the registry and any file copies made.

Although restore points can be created with the *System Restore* wizard (see Figure 3.17), it is also possible to perform the exact same operation from a script by using the *SystemRestore* class supported by the *System Restore* WMI provider. Only available under Windows XP and registered under the Root\Default namespace, the *System Restore* WMI provider is implemented as an instance and method provider (see Table 3.29).

Table 3.29

The SystemRestore Providers 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
System Restore Provider	Root/Default	X X						X X X				X				

The *System Restore* provider supports two WMI classes (see Table 3.30).

Table 3.30 The *SystemRestore Providers Classes*

Name	Type	Comments
SystemRestore	Dynamic	Provides methods for disabling and enabling monitoring, listing available restore points, and initiating a restore on the local system.
SystemRestoreConfig	Dynamic	Provides properties for controlling the frequency of scheduled restore point creation and the amount of disk space consumed on each drive.

The class supporting *System Restore* operations from a script is the *SystemRestore* class. Basically, this class supports five methods implementing the five typical *System Restore* operations. As with the wizard, it is possible to create new restore points, to enable or disable the *System Restore* monitoring for all disks or per disk, get the last *System Restore* status, and restore a specific restore point with this class. To configure the various intervals and the percentage of disk space used by *System Restore*, the *SystemRestoreConfig* class must be used. The logic making use of these classes is implemented in Samples 3.25 through 3.31. The code is written in Jscript and is called **WMI-SystemRestore.Wsf**. This sample exposes a set of command-line parameters corresponding to methods exposed by the *SystemRestore* class and properties exposed by the *SystemRestoreConfig* class.

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

Usage: WMISystemRestore.wsf /Action:value [/Volume:value] [/Description:value]
                  [/RestoreSequence[+|-]] [/DiskPercentage[+|-]] [/GlobalInterval[+|-]]
                  [/LifeInterval[+|-]] [/SessionInterval[+|-]]
                  [/Machine:value] [/User:value] [/Password:value]

Options:
Action      : Specify the operation to perform: [List], [Disable], [Enable],
              [CreateRestorePoint], [LastRestoreStatus], [Restore] and [Update].
Volume     : Specify the System Resstore volume (i.e., C:\ or * for all volumes).
Description : Description for the new restore point.
RestoreSequence : Sequence number of the restore point.
DiskPercentage : Maximum amount of disk space on each drive that can be used by System Restore.
GlobalInterval : Absolute time interval at which scheduled system checkpoints are created (hours).
LifeInterval  : Time interval for which restore points are preserved (hours).
SessionInterval : Time interval at which scheduled system checkpoints are created
                  during the session (hours)
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)

Examples:
WMISystemRestore.wsf /Action>List
WMISystemRestore.wsf /Action:Disable /Volume:D:\
WMISystemRestore.wsf /Action:Enable /Volume:D:\
WMISystemRestore.wsf /Action:Disable /Volume:*
WMISystemRestore.wsf /Action:Enable /Volume:*
```

```
WMISystemRestore.wsf /Action>CreateRestorePoint /Description:"My System Restore"
WMISystemRestore.wsf /Action>LastRestoreStatus
WMISystemRestore.wsf /Action:Restore /RestoreSequence:28 /Volume:C:
WMISystemRestore.wsf /Action:Update /DiskPercentage:12 /GlobalInterval:72
                                         /LifeInterval:120 /SessionInterval:120
```

As with other script samples, the **WMISystemRestore.Wsf** script (see Sample 3.25) starts with the command-line parameter definition (skipped lines 13 through 37), the inclusion of some external functions (lines 39 through 45), the command-line parsing (skipped lines 49 through 147), and the WMI connection (lines 148 through 159).

→ **Sample 3.25** *Exploiting the System Restore features from script (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
37:  </runtime>
38:
39:  <script language="VBScript" src=..\\Functions\\DecodeSystemRestoreFunction.vbs" />
40:
41:  <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertyFunction.vbs" />
42:  <script language="VBScript" src=..\\Functions\\TinyErrorHandler.vbs" />
43:
44:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
45:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
46:
47:  <script language="Jscript">
48:  <![CDATA[
49:  var cComputerName = "LocalHost";
50:  var cWMINameSpace = "Root/Default";
51:
.:
147:
148:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault;
149:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate;
150:
151:  try
152:  {
153:    objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace,
154:                                                   strUserID, strPassword);
155:  }
156:  catch (Err)
157:  {
158:    ErrorHandler (Err);
159:  }
160:
.:
.:
.:
```

To view the restore points available, Sample 3.26 retrieves all instances of the *SystemRestore* class (line 166). The scripting technique is quite easy, since it simply consists of an enumeration of all instances (lines 173 through 218) with all their properties (lines 183 through 216).

→ **Sample 3.26** *Viewing the System Restore points (Part II)*

```
...:  
...:  
...:  
160:  
161: // -- LIST -----  
162: if (boolList == true)  
163: {  
164:     try  
165:     {  
166:         objWMIIInstances = objWMIServices.InstancesOf ("SystemRestore")  
167:     }  
168:     catch (Err)  
169:     {  
170:         ErrorHandler (Err)  
171:     }  
172:  
173:     enumWMIIInstances = new Enumerator (objWMIIInstances);  
174:     for (;! enumWMIIInstances.atEnd(); enumWMIIInstances.moveNext())  
175:     {  
176:         objWMIIInstance = enumWMIIInstances.item();  
177:  
178:         objWMIDateTime.Value = objWMIIInstance.CreationTime;  
179:         WScript.Echo ("-- " + objWMIIInstance.Description + " (" +  
180:                         objWMIDateTime.GetVarDate (false) + ")" +  
181:                         "-----");  
182:  
183:         objWMIPropertySet = objWMIIInstance.Properties_  
184:         enumWMIPropertySet = new Enumerator (objWMIPropertySet);  
185:         for (;! enumWMIPropertySet.atEnd(); enumWMIPropertySet.moveNext())  
186:         {  
187:             objWMIProperty = enumWMIPropertySet.item()  
188:  
189:             switch (objWMIProperty.Name)  
190:             {  
191:                 case "RestorePointType":  
192:                     DisplayFormattedProperty (objWMIIInstance,  
193:                         " " + objWMIProperty.Name,  
194:                         DecodeRestorePointType (objWMIProperty.Value),  
195:                         null);  
196:                     break;  
197:                 case "EventType":  
198:                     DisplayFormattedProperty (objWMIIInstance,  
199:                         " " + objWMIProperty.Name,  
200:                         DecodeEventType (objWMIProperty.Value),  
201:                         null);  
202:                     break;  
203:                 case "CreationTime":  
204:                     DisplayFormattedProperty (objWMIIInstance,  
205:                         " " + objWMIProperty.Name,  
206:                         objWMIDateTime.GetVarDate (false),
```

```

207:                     null);
208:                 break;
209:             default:
210:                 DisplayFormattedProperty (objWMIInstance,
211:                     " " + objWMIProperty.Name,
212:                     objWMIProperty.Name,
213:                     null);
214:                 break;
215:             }
216:         }
217:         WScript.Echo();
218:     }
219: }
220:
...:
...:
...

```

To enable or disable the *System Restore* disk monitoring, the scripting technique is pretty simple. Sample 3.27 retrieves an instance of the *SystemRestore* class (lines 224 and 250). Because these *Disable* and *Enable* methods do not relate to a particular dynamic instance, they are defined as static methods (Static qualifier set to true) in the CIM repository. Actually, the logical disk for which the *System Restore* monitoring must be enabled or disabled must be specified as a parameter of the methods (lines 224 and 250). If a drive letter is specified on the command line (in the form “C:\” or “D:\”), the method will be related to the specified disk (lines 232 and 258). If an asterisk is given on the command line, the method will apply to all disks (lines 226 and 252). In this case, the disk letter will be an empty string when passing the parameter to the method (lines 228 and 254). Interestingly, the disk is not managed via a WMI instance stored in an SWBemObject, which represents the real disk. Instead, it is managed by an instance of the class where parameters specify the disk to manage (lines 228 and 254). This implementation is totally dependent on how the WMI provider and its supported classes are designed.

Sample 3.27 *Enabling/disabling disk monitoring (Part III)*

```

...:
...:
...:
220:
221: // -- ENABLE -----
222: if (boolEnable == true)
223: {
224:     objWMIClass = objWMIServices.Get ("SystemRestore")
225:
226:     if (strDeviceID == "*")
227:     {
228:         intRC = objWMIClass.Enable ("", true)
229:     }
230:     else

```

```
231:      {
232:          intRC = objWMIClass.Enable (strDeviceID, true)
233:      }
234:
235:      if (intRC)
236:      {
237:          WScript.Echo ("Failed to enable SystemRestore on " +
238:                         strDeviceID + " drive (" + intRC + ")")
239:      }
240:      else
241:      {
242:          WScript.Echo ("SystemRestore on " + strDeviceID +
243:                         " drive successfully enabled.")
244:      }
245:  }
246:
247: // -- DISABLE -----
248: if (boolDisable == true)
249: {
250:     objWMIClass = objWMIservices.Get ("SystemRestore")
251:
252:     if (strDeviceID == "*")
253:     {
254:         intRC = objWMIClass.Disable ("")
255:     }
256:     else
257:     {
258:         intRC = objWMIClass.Disable (strDeviceID)
259:     }
260:
261:     if (intRC)
262:     {
263:         WScript.Echo ("Failed to disable SystemRestore on " +
264:                         strDeviceID + " drive (" + intRC + ")")
265:     }
266:     else
267:     {
268:         WScript.Echo ("SystemRestore on " + strDeviceID +
269:                         " drive successfully disabled.")
270:     }
271: }
272:
...:
...:
...:
```

To create a restore point, the same logic applies as enabling or disabling a disk (see Sample 3.28). An instance of the *SystemRestore* class is first retrieved (line 276). Next, the *CreateRestorePoint* method is invoked (lines 278 through 280).

→ **Sample 3.28** *Creating a restore point (Part IV)*

```
...:
...:
...:
272:
273: // -- CreateRestorePoint -----
274: if (boolCreateRestorePoint == true)
275: {
```

```

276:     objWMIClass = objWMIServices.Get ("SystemRestore")
277:
278:     intRC = objWMIClass.CreateRestorePoint (strDescription,
279:                                             APPLICATION_INSTALL,
280:                                             BEGIN_SYSTEM_CHANGE)
281:
282:     if (intRC)
283:     {
284:         WScript.Echo ("Failed to create a new system restore point (" + intRC + ")")
285:     }
286:     else
287:     {
288:         WScript.Echo ("System restore point successfully created.")
289:     }
290: }
291:
...:
...:
...

```

The three parameters passed along the method invocation are as follows:

- **Description:** The *Description* parameter is a string used to identify the restore point.
- **RestorePointType:** The *RestorePointType* is an integer defining the type of restore point. The various values that can be specified with this parameter are summarized in Table 3.31.

Table 3.31 *The RestorePointType Parameter*

Name	Value	Comments
APPLICATION_INSTALL	0	An application has been installed.
APPLICATION_UNINSTALL	1	An application has been uninstalled.
DEVICE_DRIVER_INSTALL	10	A device driver has been installed.
MODIFY_SETTINGS	12	An application has had features added or removed.
CANCELLED_OPERATION	13	An application needs to delete the restore point it created. For example, an application would use this flag when a user cancels an installation.

- **EventType:** The *EventType* is an integer defining the type of event for the restore point. The various values that can be specified with this parameter are summarized in Table 3.32.

Table 3.32 *The EventType Parameter*

Name	Value	Comments
BEGIN_NESTED_SYSTEM_CHANGE	102	A system change has begun. A subsequent nested call does not create a new restore point. Subsequent calls must use END_NESTED_SYSTEM_CHANGE, not END_SYSTEM_CHANGE.
BEGIN_SYSTEM_CHANGE	100	A system change has begun.
END_NESTED_SYSTEM_CHANGE	103	A system change has ended.
END_SYSTEM_CHANGE	101	A system change has ended.

Getting the last restore status from the script is a very simple operation (see Sample 3.29). The *GetLastRestoreStatus* method is a static method and is therefore executed from an instance of the *SystemRestore* class (lines 295 through 297). The last restore status is returned as a result of the method execution.

→ **Sample 3.29** *Getting the last restore status (Part V)*

```
...:  
...:  
...:  
291:  
292: // -- LastRestoreStatus -----  
293: if (boolRestoreStatus == true)  
294: {  
295:     objWMIClass = objWMIServices.Get ("SystemRestore")  
296:  
297:     intRC = objWMIClass.GetLastRestoreStatus ()  
298:  
299:     WScript.Echo ("Last restore status: " + intRC)  
300: }  
301:  
...:  
...:  
...:
```

To restore a restore point, the coding is no more complicated than getting the last restore status (see Sample 3.30). The *Restore* static method must still be executed from an instance of *SystemRestore* class. The only required input parameter to execute the *Restore* method is the restore point sequence number. The restore point sequence number can be found by executing the **WMISystemRestore.Wsf** script with the **/Action>List** command-line parameter. A sample output will look as follows:

```
1: C:\>WMISystemRestore.wsf /Action>List  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5: - System Checkpoint (Fri Oct 11 10:33:16 UTC+0200 2002)-----  
6: CreationTime: ..... 11-10-2002 10:33:16  
7: Description: ..... System Checkpoint  
8: EventType: ..... BEGIN_SYSTEM_CHANGE  
9: RestorePointType: ..... APPLICATION_INSTALL  
10: *SequenceNumber: ..... 45  
11:  
12: - Windows Update V4 (Fri Oct 11 21:38:54 UTC+0200 2002)-----  
13: CreationTime: ..... 11-10-2002 21:38:54  
14: Description: ..... Windows Update V4  
15: EventType: ..... BEGIN_SYSTEM_CHANGE  
16: RestorePointType: ..... APPLICATION_INSTALL  
17: *SequenceNumber: ..... 46  
18:
```

```

19: - System Checkpoint (Sun Oct 13 07:50:20 UTC+0200 2002)-----
20: CreationTime: ..... 13-10-2002 07:50:20
21: Description: ..... System Checkpoint
22: EventType: ..... BEGIN_SYSTEM_CHANGE
23: RestorePointType: ..... APPLICATION_INSTALL
24: *SequenceNumber: ..... 47

```

As we can see, the sequence number is a key property of the *SystemRestore* instances (lines 10, 17, and 24).

Sample 3.30 *Restoring a restore point (Part VI)*

```

...:
...:
...:
301:
302: // -- Restore -----
303: if (boolCreateRestorePoint == true)
304: {
305:     objWMIClass = objWMIServices.Get ("SystemRestore")
306:
307:     intRC = objWMIClass.Restore(intSequenceNumber)
308:
309:     if (intRC)
310:     {
311:         WScript.Echo ("Failed to create a new system restore point (" + intRC + ")")
312:     }
313:     else
314:     {
315:         WScript.Echo ("System restore point successfully created.")
316:     }
317: }
318:
...:
...:
...:

```

To manage the properties controlling the frequency of the scheduled restore point creation and the amount of disk space consumed on each drive, the *SystemRestoreConfig* class must be used, as shown in Sample 3.31.

Sample 3.31 *Updating the System Restore parameters (Part VII)*

```

...:
...:
...:
318:
319: // -- Update -----
320: if (boolUpdate == true)
321: {
322:     objWMIInstance = objWMIServices.Get ("SystemRestoreConfig.MyKey='SR'");
323:
324:     if (intDiskPercentage != -1) objWMIInstance.DiskPercent=intDiskPercentage;
325:     if (intGlobalInterval != -1) objWMIInstance.RPGlobalInterval=intGlobalInterval*3600;

```

```

326:     if (intLifeInterval != -1) objWMIInstance.RPLifeInterval=intLifeInterval*3600;
327:     if (intSessionInterval != -1) objWMIInstance.RPSessionInterval=intSessionInterval*3600;
328:
329:     try
330:     {
331:         objWMIInstance.Put_ (wbemChangeFlagUpdateOnly | wbemFlagReturnWhenComplete);
332:     }
333:     catch (Err)
334:     {
335:         ErrorHandler (Err);
336:     }
337:
338:     WScript.Echo ("System restore point parameters updated.")
339: }
340: ]]
341: </script>
342: </job>
343:</package>
```

The only instance available from the *SystemRestoreConfig* class is the SR instance (line 322). That instance exposes the disk percentage in the *DiskPercent* property and the three frequency properties in *RPGlobalFrequency*, *RPLifeInterval*, and *RPSessionInterval*, respectively (lines 324 through 327). The meaning of these properties is summarized in Table 3.33. Note that the script accepts the schedule parameters in hours, while the *SystemRestoreConfig* class exposes these properties in seconds. That's why the script code multiplies the values by 3,600 during the property assignment (lines 324 through 327).

Table 3.33 The *SystemRestoreConfig* Properties

Name	Comments
DiskPercent	Maximum amount of disk space on each drive that can be used by System Restore. This value is specified as a percentage of the total drive space. The default value is 12 percent.
RPGlobalInterval	Absolute time interval at which scheduled system checkpoints are created, in seconds. The default value is 86,400 (24 hours).
RPLifeInterval	Time interval for which restore points are preserved, in seconds. When a restore point becomes older than this specified interval, it is deleted. The default age limit is 90 days.
RPSessionInterval	Time interval at which scheduled system checkpoints are created during the session, in seconds. The default value is zero, indicating that the feature is turned off.

3.4 Core OS components event providers

3.4.1 The Clock provider

The *Win32_Clock* provider is an instance and an event provider. The provider capabilities are summarized in Table 3.34.

Table 3.34 The Win32ClockProvider Providers 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 XP	Windows Server 2003	Windows 2000 Professional	Windows 2000 Server
Clock Provider															
Win32ClockProvider	Root/CIMV2	X	X	X	X	X	X	X	X	X	X	X	X	X	X

As shown in Table 3.35, this provider supports two classes: the *Win32_LocalTime* and *Win32_UTCTime* classes.

Table 3.35 The Win32ClockProvider Classes

Name	Type	Comments
Win32_LocalTime	Dynamic (Singleton)	Represents an instance of the local time
Win32_UTCTime	Dynamic (Singleton)	Represents an instance of the UTC time

These two classes are created from the *Win32_CurrentTime* superclass (see Figure 3.18). All classes are singleton classes. There is no particular event class, since the *Clock* provider works with the *__InstanceModification-Event* intrinsic event class.

Figure 3.18
The Win32_CurrentTime class and its child classes.



Because this provider is implemented as an event provider, it is possible to formulate a WQL query without the WITHIN statement. For example, the following query:

```
Select * From __InstanceModificationEvent Where TargetInstance ISA 'Win32_LocalTime'
```

will trigger a notification every time the local time changes. We can obtain the same result by performing a WQL event query with the *Win32_UTCTime*:

```
Select * From __InstanceModificationEvent Where TargetInstance ISA 'Win32_UTCTime'
```

Now, if we want to get a notification every new minute for the UTC time, we can use the following query:

```
Select * From __InstanceModificationEvent Where TargetInstance ISA 'Win32_UTCTime' AND  
TargetInstance.Second=0
```

In *Understanding WMI Scripting*, Chapter 6, when we talked about the Timer Events, we saw how to use the interval timer event with its corresponding *__IntervalTimerInstruction* class. This last WQL event query could represent a good alternative to the interval timer event.

With the help of the *Win32_CurrentTime* class, it is possible to get the current system time. In the previous chapter, we wrote a script to manage scheduled jobs. This script makes use of the *Win32_CurrentTime* class, but we didn't examine this part of the code (see Samples 2.56 through 2.59). When we schedule jobs, it is sometimes useful to retrieve the current system time, especially when these jobs are scheduled on a remote computer. Because the *Win32_CurrentTime* class is a superclass for the *Win32_UTCTime* and *Win32_LocalTime* classes, it is possible to retrieve the current time in two forms: UTC and localized.

The portion of code retrieving the system time is shown in Sample 3.32 (lines 275 and 319) and is an extract of Samples 2.56 through 2.59.

→ **Sample 3.32** *Getting the current time (UTC and local)*

```
1:<?xml version="1.0"?>  
..:  
8:<package>  
9:  <job>  
...:  
13:   <runtime>  
...:  
37:   </runtime>  
38:  
39:   <script language="VBScript" src=".\\Functions\\DecodeDaysOfWeekFunction.vbs" />  
40:   <script language="VBScript" src=".\\Functions\\DecodeDaysOfMonthFunction.vbs" />  
41:  
42:   <script language="VBScript" src=".\\Functions\\ConvertStringInArrayFunction.vbs" />  
43:   <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />  
44:   <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />  
45:  
46:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>  
47:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />  
48:  
49:   <script language="VBScript">  
50:     <![CDATA[  
..:  
54:     '  
55:     Const cComputerName = "LocalHost"  
56:     Const cWMINNameSpace = "Root\\cimv2"
```

```
57: Const cWMIScheduledJobClass = "Win32_ScheduledJob"
58: Const cWMICurrentTimeClass = "Win32_CurrentTime"
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74: ' TIME -----
75: If boolGetTime Then
76:     Set objWMInstances = objWMIServices.InstancesOf (cWMICurrentTimeClass)
77:
78:
79:     For Each objWMInstance in objWMInstances
80:         objWMIDateTime.Year = objWMInstance.Year
81:         objWMIDateTime.YearSpecified = True
82:         objWMIDateTime.Month = objWMInstance.Month
83:         objWMIDateTime.MonthSpecified = True
84:         objWMIDateTime.Day = objWMInstance.Day
85:         objWMIDateTime.DaySpecified = True
86:
87:         objWMIDateTime.Hours = objWMInstance.Hour
88:         objWMIDateTime.HoursSpecified = True
89:         objWMIDateTime.Minutes = objWMInstance.Minute
90:         objWMIDateTime.MinutesSpecified = True
91:         objWMIDateTime.Seconds = objWMInstance.Second
92:         objWMIDateTime.SecondsSpecified = True
93:
94:         objWMIDateTime.IsInterval = False
95:         If objWMInstance.Path_.Class = "Win32_UTCTime" Then
96:             WScript.Echo "- UTC " & String (70, "-")
97:             WScript.Echo "Current date/time is: " & _
98:                 objWMIDateTime.GetVarDate (False) & _
99:                     " (" & objWMIDateTime.Value & ")."
100:
101:     Else
102:         WScript.Echo "- Local " & String (68, "-")
103:         WScript.Echo "Current date/time is: " & _
104:             objWMIDateTime.GetVarDate (False) & _
105:                 " (" & objWMIDateTime.Value & ")."
106:
107:
108:     Set objWMIPropertySet = objWMInstance.Properties_
109:     For Each objWMIProperty In objWMIPropertySet
110:         DisplayFormattedProperty objWMInstance, _
111:             objWMIProperty.Name, _
112:             objWMIProperty.Name, _
113:             Null
114:
115:         Next
116:
117:
118:     Next
119:     WScript.Echo
120:
121:
```

```
324:  1]>
325:  </script>
326:  </job>
327:</package>
```

Because we want the script to show the time type (UTC or local), it retrieves the collection available from the *Win32_CurrentTime* (line 276). This collection is made up of the *Win32_UTCTime* singleton instance and the *Win32_LocalTime* singleton instance. Next, it enumerates the collection (lines 279 through 317), and, for each instance found, the script stores the time result in an *SWBemDateTime* object (lines 280 through 292). Based on the class name of the retrieved time (*Win32_LocalTime* or *Win32_UTCTime* at line 295), the script displays the corresponding time message with the class properties. Once executed, we get the following output:

```
1: C:\>WMIScheduledJob.wsf /GetTime+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Local -----
6: Current date/time is: 01-11-2001 19:35:54 (20011101193554.000000+000).
7: Day: ..... 1
8: DayOfWeek: ..... 4
9: Hour: ..... 19
10: Minute: ..... 35
11: Month: ..... 11
12: Quarter: ..... 4
13: Second: ..... 54
14: WeekInMonth: ..... 1
15: Year: ..... 2001
16:
17: - UTC -----
18: Current date/time is: 01-11-2001 18:35:54 (20011101183554.000000+000).
19: Day: ..... 1
20: DayOfWeek: ..... 4
21: Hour: ..... 18
22: Minute: ..... 35
23: Month: ..... 11
24: Quarter: ..... 4
25: Second: ..... 54
26: WeekInMonth: ..... 1
27: Year: ..... 2001
```

3.4.2 Power management provider

The *power management* provider consists of only one event provider (see Table 3.36) supporting one event class.

This provider is designed to trigger a WMI event notification to every event consumer who has subscribed to receive power management event notifications. The *Win32_PowerManagementEvent* class is the only class supported and is an extrinsic event class available in the *Root\CIMv2*

Table 3.36 The Power Management Providers 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
Power Management Provider MS_Power_Management_Event_Provider	Root/CIMV2					X						X	X	X	X	X

namespace. This provider is not an instance or property provider and therefore does not expose information about the power devices themselves. To gather information about the power devices, you must refer to the previous chapter (section 2.3.5), and work with one of the following classes: *Win32_Battery*, *Win32_CurrentProbe*, *Win32_PortableBattery*, *Win32_UninterruptiblePowerSupply*, or *Win32_VoltageProbe*. The WQL event query to receive all power management events is as follows:

```
Select * From Win32_PowerManagementEvent
```

The easiest way to test this WQL event query with a script is to execute Sample 6.17 (“A generic script for asynchronous event notification”), available in the appendix on a laptop. For example, once you have started the script with the following command line, you can switch the laptop to standby mode. Note that if you switch your laptop to hibernate mode, you should obtain the same result. The output would be as follows:

```

1: C:\>GenericEventAsyncConsumer.wsf "Select * From Win32_PowerManagementEvent"
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Tuesday, 20 November, 2001 at 15:41:27: 'Win32_PowerManagementEvent' has been triggered.
9:   EventType (wbemCimtypeUInt16) = 4
10:  OEMEventCode (wbemCimtypeUInt16) = (null)
11: END - OnObjectReady.
12:
13: BEGIN - OnObjectReady.
14: Tuesday, 20 November, 2001 at 15:41:56: 'Win32_PowerManagementEvent' has been triggered.
15:   EventType (wbemCimtypeUInt16) = 18
16:  OEMEventCode (wbemCimtypeUInt16) = (null)
17: END - OnObjectReady.
18:
19: BEGIN - OnObjectReady.
20: Tuesday, 20 November, 2001 at 15:41:56: 'Win32_PowerManagementEvent' has been triggered.
21:   EventType (wbemCimtypeUInt16) = 7
22:  OEMEventCode (wbemCimtypeUInt16) = (null)
23: END - OnObjectReady.
```

Each time a power management event occurs (lines 7, 13, and 19), the script receives the event represented by a *Win32_PowerManagementEvent* instance. This class exposes a property called *EventType*, and its value corresponds to the power management event type (lines 9, 15, and 21). The meaning of the values is shown in Table 3.37.

Table 3.37

The Power Management Event Type Values

Meaning	Values
Entering Suspend	4
Resume from Suspend	7
Power Status Change	10
OEM Event	11
Resume Automatic	18

In the sample output, the *OEMEventCode* property is always set to Null, because there is no OEM event reported. Note that if you remove the power supply of the laptop, it will switch on the battery and this will trigger power management event 10 (“Power Status Change”).

This event type can be useful for applications that must perform specific tasks when the power status of the computer changes.

3.4.3 Shutdown provider

As with the *power management* provider, the *shutdown* provider is also made up of one event provider supporting only one single extrinsic event class, which is called *Win32_ComputerShutdownEvent* (see Table 3.38.)

Table 3.38

The Shutdown Providers 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
Shutdown Provider																
MS Shutdown Event Provider	Root\CLIMV2		X									X	X			

This event class represents events when a computer has begun the process of shutting down. To receive a computer shutdown notification, the following WQL event query must be used:

```
Select * From Win32_ComputerShutdownEvent
```

Again by reusing Sample 6.17 ("A generic script for asynchronous event notification") available in the appendix, we obtain the following output:

```
1: C:\>GenericEventAsyncConsumer.wsf "Select * From Win32_ComputerShutdownEvent"
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Tuesday, 20 November, 2001 at 16:46:33: 'Win32_ComputerShutdownEvent' has been triggered.
9:
10: - Win32_ComputerShutdownEvent -----
11: MachineName: ..... NET-DPEN6400A
12: TIME_CREATED: ..... 20-11-2001 14:44:22 (20011120134422.849164+060)
13: Type: ..... 0
14:
15: END - OnObjectReady.
16:
17: BEGIN - OnObjectReady.
18: Tuesday, 20 November, 2001 at 16:46:39: 'Win32_ComputerShutdownEvent' has been triggered.
19:
20: - Win32_ComputerShutdownEvent -----
21: MachineName: ..... NET-DPEN6400A
22: TIME_CREATED: ..... 20-11-2001 14:44:28 (20011120134428.717603+060)
23: Type: ..... 1
24:
25:
```

The output sample is obtained when a server reboot or shutdown is requested. You will notice two events: the first event (lines 11 through 13) corresponds to a Logoff (value 0 of the *type* property at line 13); the second event (lines 21 through 23) corresponds to a shutdown or reboot (value 1 of the *type* property at line 23). From this output, it is interesting to note that this provider notifies any Logoff event to the subscribed consumers in addition to detecting Operating System shutdowns.

As with the power management event, this event type can be useful for an application that must perform some specific tasks when a user logoff or machine shutdown is invoked.

3.4.4 Configuration Change provider

The *Configuration Change* provider is implemented as an event provider (Table 3.39). Only available under Windows XP or Windows Server 2003, this provider indicates with the *Win32_SystemConfigurationChangeEvent* extrinsic event class that the device list on the system has been refreshed. This means that a device has been added, removed, or reconfigured.

Table 3.39 The Configuration Change Providers 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
Configuration Change Provider	SystemConfigurationChangeEvents	Root/CIMV2			X							X	X			

The *Win32_SystemConfigurationChangeEvent* event class is the only class supported by the *Configuration Change* provider. The change to the device list is not contained in the event and therefore an application or a script is required to refresh its knowledge of the device list in order to obtain the current system settings. Configuration changes can be anything related to the system configuration, such as IRQ settings, COM ports, and BIOS version, to name a few. For example, in the previous chapter, we developed a script to retrieve the hardware resource information (see Samples 2.4 through 2.7, “Retrieving hardware resource information”). This script can be easily reused and expanded to determine the updated configuration. The only relevant information contained in the extrinsic event is the event type, which is contained in the *EventType* property. This property indicates the type of device change notification event that has occurred (Table 3.40).

Table 3.40 The EventType Property Meaning

Meaning	Values
Configuration Changed	1
Device Arrival	2
Device Removal	3
Docking	4

If we reuse Sample 6.17 (“A generic script for asynchronous event notification”) in the appendix, and if you connect a USB device to your Windows Server 2003 or Windows XP system, you may get an output similar to the following one:

```

1: C:\>GenericEventAsyncConsumer.wsf "Select * From Win32_SystemConfigurationChangeEvent"
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Sunday, 17 Feb, 2002 at 11:01:00: 'Win32_SystemConfigurationChangeEvent' has been triggered.

```

```

9:     EventType (wbemCimtypeUInt16) = 1
10:    SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
11:    TIME_CREATED (wbemCimtypeUInt64) = 17-02-2002 10:01:00 (20020217100100.359920+060)
12:
13: END - OnObjectReady.
14:
15: BEGIN - OnObjectReady.
16: Sunday, 17 Feb, 2002 at 11:01:00: 'Win32_SystemConfigurationChangeEvent' has been triggered.
17:     EventType (wbemCimtypeUInt16) = 1
18:     SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
19:     TIME_CREATED (wbemCimtypeUInt64) = 17-02-2002 10:01:00 (20020217100100.460064+060)
20:
21: END - OnObjectReady.

```

Based on Table 3.40, we clearly see that the WMI event corresponds to a configuration change (lines 9 and 17).

3.4.5 Volume Change event provider

The *Volume Change* event provider supports only one extrinsic event class available in the **Root\CIMv2** namespace (Table 3.41). Its purpose is to detect the addition or the removal of a drive letter or mounted/dismounted drive on the computer system.

Table 3.41 *The Volume Change Providers 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
Volume Change Provider																
VolumeChangeEvents	Root/CIMV2				X							X	X			

The *Win32_VolumeChangeEvent* event class represents a local drive event resulting from the change. Network drives are not currently supported. The *Win32_VolumeChangeEvent* event class is generally used in a WQL event query:

```
Select * From Win32_VolumeChangeEvent
```

If we reuse Sample 6.17 (“A generic script for asynchronous event notification”) in the appendix, and if we change the drive letter of volume E: to Z: in a Windows Server 2003 or Windows XP system, we may get an output similar to the following one:

```
C:\>GenericEventAsyncConsumer.wsf "Select * From Win32_VolumeChangeEvent"
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Waiting for events...

BEGIN - OnObjectReady.
Wednesday, 14 August, 2002 at 17:23:56: 'Win32_VolumeChangeEvent' has been triggered.
  DriveName (wbemCimtypeString) = E:
  EventType (wbemCimtypeUInt16) = 3
  SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
  TIME_CREATED (wbemCimtypeUInt64) = 14-08-2002 15:23:56 (20020814152356.796875+120)

END - OnObjectReady.

BEGIN - OnObjectReady.
Wednesday, 14 August, 2002 at 17:23:57: 'Win32_VolumeChangeEvent' has been triggered.
  DriveName (wbemCimtypeString) = Z:
  EventType (wbemCimtypeUInt16) = 2
  SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
  TIME_CREATED (wbemCimtypeUInt64) = 14-08-2002 15:23:56 (20020814152356.984375+120)

END - OnObjectReady.
```

The *Win32_VolumeChangeEvent* class exposes an *EventType* property. You can refer to Table 3.40 for more information about this property.

3.5 Core OS file system components providers

3.5.1 Disk quota provider

The *Disk quota* provider is made up of only one instance provider (see Table 3.42) supporting three classes. This provider is designed to expose information about quota settings configured on NTFS volumes.

Table 3.42 The Disk Quota Providers 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
Disk Quota Provider																
DiskQuotaProvider	Root/CIMV2	X						X	X	X	X	X	X			

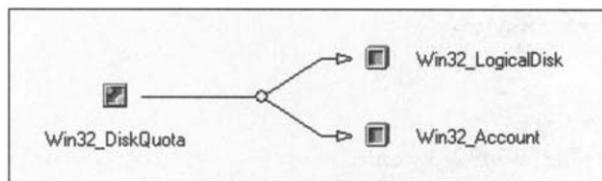
It supports one dynamic instance class and two association classes, as shown in Table 3.43.

Table 3.43 *The Disk Quota Provider Classes*

Name	Type	Comments
Win32_QuotaSetting	Dynamic	Contains setting information for disk quotas on a volume.
Win32_DiskQuota	Association	Tracks disk space usage for NTFS volumes.
Win32_VolumeQuotaSetting	Association	Relates disk quota settings with a specific disk volume.

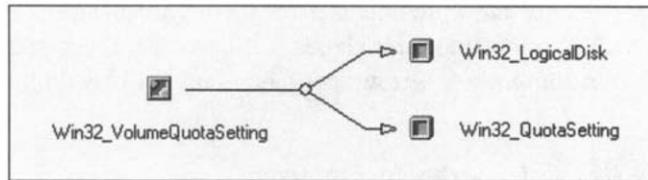
These classes are available in the Root\CMV2 namespace. The *Win32_DiskQuota* association class associates the *Win32_LogicalDisk* class with the *Win32_Account* (see Figure 3.19).

Figure 3.19
The Win32_DiskQuota association class.



With this association, it is possible to view and configure different quotas per user. We will exploit this capability in the next script sample. On the other hand, the *Win32_QuotaSetting* class is associated with the *Win32_LogicalDisk* class via the *Win32_VolumeQuotaSetting* class (Figure 3.20).

Figure 3.20
The Win32_QuotaSetting class associations.



With this association, it is possible to view and configure the default quota settings for each NTFS volume. In the previous chapter, when we examined the File System, we saw how to use the *Win32_LogicalDisk* class. To illustrate its use, we developed Samples 2.31 through 2.34 (“Gathering disk partition, disk drive, and logical disk information”). However, we skipped some lines (lines 245 through 267), because this piece of code was related to the disk quota information. These lines are presented in Sample 3.33.

Sample 3.33 Retrieving Disk quota information for each logical disk

```
...:  
...:  
...:  
245:  
246:     Set objWMIQuotaInstances = objWMIServices.ExecQuery _  
247:             ("Associators of (Win32_LogicalDisk=''" & _  
248:                 objWMILogicalDiskInstance.DeviceID & _  
249:                 '') Where AssocClass=Win32_VolumeQuotaSetting")  
250:  
251:     If objWMIQuotaInstances.Count Then  
252:         WScript.Echo  
253:         WScript.Echo " -- Quota information " & " " & String (63, "-")  
254:         For Each objWMIQuotaInstance In objWMIQuotaInstances  
255:             Set objWMIPropertySet = objWMIQuotaInstance.Properties_  
256:             For Each objWMIProperty In objWMIPropertySet  
257:                 DisplayFormattedProperty objWMIQuotaInstance, _  
258:                     " " & objWMIProperty.Name, _  
259:                     objWMIProperty.Name, _  
260:                     Null  
261:             Next  
262:             Set objWMIPropertySet = Nothing  
263:         Next  
264:     End If  
265:  
266:     Set objWMIQuotaInstances = Nothing  
267:  
...:  
...:  
...:
```

In this sample, we use the association class *Win32_VolumeQuotaSetting* to retrieve the *Win32_QuotaSetting* instances associated with the *Win32_LogicalDisk* (lines 246 through 249). Once the collection of instances is available, the script displays the disk quota information (lines 253 through 263). The quota information is available between lines 70 and 77 in the following output:

```
1: C:\>GetPartitionInformation.wsf  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5: - Disk #0, Partition #0 -----  
6:   BlockSize: ..... 512  
7:   Bootable: ..... TRUE  
...  
15:   PrimaryPartition: ..... TRUE  
16:   Size: ..... 843816960  
17:   StartingOffset: ..... 874782720  
18:   Type: ..... MS-DOS V4 Huge  
19:  
20: -- Physical disk information -----  
21:   BytesPerSector: ..... 512  
22:   Capabilities: ..... Random Access, Supports Writing  
...:
```

```

44: TotalCylinders: ..... 555
45: TotalHeads: ..... 240
46: TotalSectors: ..... 8391600
47: TotalTracks: ..... 133200
48: TracksPerCylinder: ..... 240
49:
50: -- Logical disk information -----
51: Compressed: ..... FALSE
52: Description: ..... Local Fixed Disk
...
60: QuotasDisabled: ..... FALSE
61: QuotasIncomplete: ..... FALSE
62: QuotasRebuilding: ..... FALSE
63: Size: ..... 843816448
64: SupportsDiskQuotas: ..... TRUE
65: SupportsFileBasedCompression: ..... TRUE
66: VolumeDirty: ..... FALSE
67: VolumeName: ..... Whistler
68: VolumeSerialNumber: ..... 988BD271
69:
70: -- Quota information -----
71: Caption: ..... C:
72: DefaultLimit: ..... 1073741824
73: DefaultWarningLimit: ..... 134217728
74: ExceededNotification: ..... TRUE
75: State: ..... 1
76: *VolumePath: ..... C:\
77: WarningExceededNotification: ..... FALSE

```

As mentioned previously, it is possible to configure a default quota per volume and a quota per user. To configure a default quota per volume, we must work with the *Win32_QuotaSetting* class. To configure a quota per user, we must work with the *Win32_DiskQuota* and exploit the associations in place (see Figure 3.19). Let's start with the default quota per volume first! The command-line parameters of the next script sample are as follows:

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

Usage: WMIQuotaSetting.wsf /Action:value /Volume:value [/DefaultLimit:value]
           [/DefaultWarningLimit[+|-]] [/ExceededNotification[+|-]]
           [/WarningExceededNotification[+|-]]
           [/Machine:value] [/User:value] [/Password:value]

Options:

Action          : Specify the operation to perform: [List], [Disabled],
                  [Tracked] and [Enforced].
Volume         : Set the volume associated with the quota.
DefaultLimit   : Set the default limit for the quota in MB.
DefaultWarningLimit : Set the default warning limit for the quota in MB.
ExceededNotification : Log event when a user exceeds quota limit.
WarningExceededNotification : Log event when a user exceeds warning level.
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)

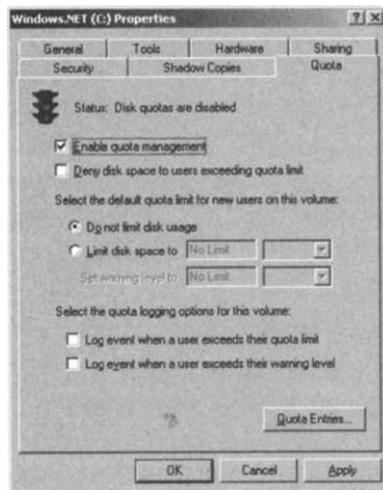
Examples:

```
WMIQuotaSetting.wsf /Action>List
WMIQuotaSetting.wsf /Volume:C:/Action:Disabled
WMIQuotaSetting.wsf /Volume:C:/Action:Tracked
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:128 /DefaultLimit:256
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:256 /DefaultLimit:NoLimit
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:NoLimit /DefaultLimit:512
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:128 /DefaultLimit:256
/ExceededNotification+
WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:128 /DefaultLimit:256
/WarningExceededNotification+
```

Each parameter exposed by the script corresponds exactly to the settings exposed by the Windows Explorer graphical interface to manage the default quota settings, as shown in Figure 3.21. By using the script, the configuration of Figure 3.21 can be obtained with the following command line:

```
C:\>WMIQuotaSetting.wsf /Volume:C:/Action:Tracked
```

Figure 3.21
The Windows
Explorer quota
management
interface.



To configure all settings available, the following command line can be used:

```
C:\>WMIQuotaSetting.wsf /Volume:C:/Action:Enforced /DefaultWarningLimit:128 /DefaultLimit:256
/ExceededNotification+ /WarningExceededNotification+
```

This command line will enforce the quota settings to a maximum default limit of 256 MB with a default warning limit at 128 MB. Each time a user uses a quota higher than the warning limit or the maximum limit, an event

log record will be created in the system NT Event Log. Let's see how to script in Jscript the default quota configuration with Samples 3.34 and 3.35.

→ **Sample 3.34** *Viewing the default volume quotas (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:    <runtime>
.:
36:    </runtime>
37:
38:    <script language="VBScript" src=..\\Functions\\DecodeVolumeQuotaStatusFunction.vbs" />
39:
40:    <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertyFunction.vbs" />
41:    <script language="VBScript" src=..\\Functions\\TinyErrorHandler.vbs" />
42:
43:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
44:
45:    <script language="Jscript">
46:      <![CDATA[
47:        var cComputerName = "LocalHost";
48:        var cWMINameSpace = "Root/cimv2";
49:        var cWMIQuotaSettingClass = "Win32_QuotaSetting";
50:
51:        var cQuotaNoLimit = "18446744073709551615";
.:
77:      // -----
78:      // Parse the command line parameters
79:      if ((WScript.Arguments.Named.Count == 0) || (WScript.Arguments.Named("Action") == null))
80:      {
81:        WScript.Arguments.ShowUsage();
82:        WScript.Quit();
83:      }
.:
112:      varDefaultLimit = WScript.Arguments.Named("DefaultLimit");
113:      if (varDefaultLimit != null)
114:        if (varDefaultLimit.toUpperCase() == "NOLIMIT")
115:        {
116:          varDefaultLimit = cQuotaNoLimit;
117:        }
118:      else
119:      {
120:        varDefaultLimit = varDefaultLimit.valueOf() * 1024 * 1024;
121:      }
122:
123:      varDefaultWarningLimit = WScript.Arguments.Named("DefaultWarningLimit");
124:      if (varDefaultWarningLimit != null)
125:        if (varDefaultWarningLimit.toUpperCase() == "NOLIMIT")
126:        {
127:          varDefaultWarningLimit = cQuotaNoLimit;
128:        }
129:      else
130:      {
131:        varDefaultWarningLimit = varDefaultWarningLimit.valueOf() * 1024 * 1024;
132:      }
```

```
...:  
154:  
155:     objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault;  
156:     objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate;  
157:  
158:     try  
159:     {  
160:         objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace,  
161:                                         strUserID, strPassword);  
162:     }  
...:  
168: // -- LIST -----  
169: if (boolList == true)  
170: {  
171:     try  
172:     {  
173:         objWMIInstances = objWMIServices.InstancesOf (cWMIQuotaSettingClass)  
174:     }  
...:  
180: enumWMIInstances = new Enumerator (objWMIInstances);  
181: for (;! enumWMIInstances.atEnd(); enumWMIInstances.moveNext())  
182: {  
183:     objWMIInstance = enumWMIInstances.item();  
184:  
185:     WScript.Echo ("-- " + objWMIInstance.Caption + " " +  
186:                 "-----");  
187:  
188:     objWMIPropertySet = objWMIInstance.Properties_  
189:     enumWMIPropertySet = new Enumerator (objWMIPropertySet);  
190:     for (;! enumWMIPropertySet.atEnd(); enumWMIPropertySet.moveNext())  
191:     {  
192:         objWMIProperty = enumWMIPropertySet.item()  
193:  
194:         switch (objWMIProperty.Name)  
195:         {  
196:             case "State":  
197:                 DisplayFormattedProperty (objWMIInstance,  
198:                                         " " + objWMIProperty.Name,  
199:                                         VolumeQuotaStatus (objWMIInstance.State),  
200:                                         null);  
201:                 break;  
202:             case "DefaultLimit":  
203:                 if (objWMIInstance.DefaultLimit == cQuotaNoLimit)  
204:                     {  
205:                         varDefaultLimit = "No Limit";  
206:                     }  
207:                 else  
208:                     {  
209:                         varDefaultLimit = objWMIInstance.DefaultLimit;  
210:                     }  
211:                 DisplayFormattedProperty (objWMIInstance,  
212:                                         " DefaultLimit (bytes)",  
213:                                         varDefaultLimit,  
214:                                         null);  
215:                 break;  
216:             case "DefaultWarningLimit":  
217:                 if (objWMIInstance.DefaultWarningLimit == cQuotaNoLimit)  
218:                     {  
219:                         varDefaultWarningLimit = "No Limit";  
220:                     }
```

```
221:             else
222:                 {
223:                     varDefaultWarningLimit = objWMIInstance.DefaultWarningLimit;
224:                 }
225:                 DisplayFormattedProperty (objWMIInstance,
226:                                         "    DefaultWarningLimit (bytes)",
227:                                         varDefaultWarningLimit,
228:                                         null);
229:             break;
230:         default:
231:             DisplayFormattedProperty (objWMIInstance,
232:                                         "    " + objWMIProperty.Name,
233:                                         objWMIProperty.Name,
234:                                         null);
235:             break;
236:         }
237:     }
238:     WScript.Echo();
239:   }
240: }
241:
...:
...:
...:
```

Once the command-line parameter definition (lines 13 through 36) and parsing (lines 77 through 154) are completed, followed by the WMI connection (lines 155 through 162), the first portion of code allows the display of the current default quota settings (lines 169 through 240). Although the Jscript language is used in this example, you will easily recognize the traditional structure to display all available instances (lines 180 through 239) with their properties (lines 188 through 237). Every volume able to support disk quotas will be displayed. We will obtain the same information as obtained from Samples 2.31 through 2.34 but without any related disk, partition, or volume information, since we are not working with the associations in this example. The most interesting part of this sample concerns the scripting logic used to enable the disk quotas and to set the miscellaneous settings. This portion of the code is shown in Sample 3.35.

Sample 3.35 Configuring the default volume quotas (Part II)

```
....  
....  
....  
241:  
242: // -- UPDATE -----  
243: if (intState > 0)  
244: {  
245:     try  
246:     {  
247:         objWMIInstance = objWMIServices.Get ("Win32_QuotaSetting.VolumePath=' " +  
248:                                         strDeviceID + "\\"");  
249:     }  
250: }
```

```
...:  
255:     if (varDefaultLimit == null)  
256:     {  
257:         varDefaultLimit = objWMIInstance.DefaultLimit;  
258:     }  
259: else  
260: {  
261:     objWMIInstance.DefaultLimit = varDefaultLimit;  
262: }  
263:  
264: if (varDefaultWarningLimit == null)  
265: {  
266:     varDefaultWarningLimit = objWMIInstance.DefaultWarningLimit;  
267: }  
268: else  
269: {  
270:     objWMIInstance.DefaultWarningLimit = varDefaultWarningLimit;  
271: }  
272:  
273: if (boolExceededNotification == null)  
274: {  
275:     boolExceededNotification = objWMIInstance.ExceededNotification;  
276: }  
277: else  
278: {  
279:     objWMIInstance.ExceededNotification = boolExceededNotification;  
280: }  
281:  
282: if (boolWarningExceededNotification == null)  
283: {  
284:     boolWarningExceededNotification = objWMIInstance.WarningExceededNotification;  
285: }  
286: else  
287: {  
288:     objWMIInstance.WarningExceededNotification = boolWarningExceededNotification;  
289: }  
290:  
291: objWMIInstance.State = (intState - 1);  
292:  
293: try  
294: {  
295:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly | wbemFlagReturnWhenComplete);  
296: }  
...:  
302: if (varDefaultLimit == cQuotaNoLimit)  
303: {  
304:     varDefaultLimit = "No Limit";  
305: }  
306: if (varDefaultWarningLimit == cQuotaNoLimit)  
307: {  
308:     varDefaultWarningLimit = "No Limit";  
309: }  
310:  
311: WScript.Echo ("Default quota setting on '" + strDeviceID + "' updated.");  
312: WScript.Echo ("Quota is '" + VolumeQuotaStatus (objWMIInstance.State) + "' (" +  
313:                 varDefaultWarningLimit + " (bytes) / " +  
314:                 varDefaultLimit + " (bytes)).");  
315: }  
316:  
317: ]]>
```

```
318:    </script>
319:  </job>
320:</package>
```

Because NTFS volumes are the only ones supporting quotas, a *Win32_QuotaSetting* instance is always available when the selected volume is an NTFS volume. Therefore, there is no need to create a *Win32_QuotaSetting* instance for a volume. However, the script must be able to update that *Win32_QuotaSetting* instance accordingly with the parameters given on the command line. This means that if only some parameters are given, other values must not be destroyed. For example, if the following command line is given:

```
1:  C:\>WMIQuotaSetting.wsf /Volume:C: /Action:Enforced /DefaultWarningLimit:256
   /WarningExceededNotification+
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Default quota setting on 'C:' updated.
6:  Quota is 'Enforced' (268435456 (bytes) / No Limit (bytes)).
```

The warning limit will be fixed to 256 MB, and an NT Event Log trace will be created once this warning limit is reached. However, the hard limit is not specified on the command line and, as Figure 3.21 shows, there is no hard limit configured. The script takes care of this missing parameter and does not change its existing value. This logic is implemented by testing the miscellaneous variables assigned by the command-line parameters. If the command-line parameter was given, the variable contains a value; otherwise, the variable is equal to Null (see lines 255, 264, 273, and 282). If the variable is Null, the current property instance value is assigned to the variable (lines 257, 266, 275, and 284). If the variable has a value, the new value is assigned to the corresponding property instance (lines 261, 270, 279, and 288).

It is important to note that when no quota limit is set the assigned value is equal to 18446744073709551615 ($2^{64} - 1$). This value is defined in the script header at line 51. If the keyword “NoLimit” is assigned, the command-line parsing code assigns the correct value to reflect the “No Limit” configuration setting (lines 112 through 132). The WMI *state* property (line 291) determines if the quota management must be disabled, tracked, or enforced. Each of these states corresponds to a value of the *state* property: 0=Disabled, 1=Tracked, and 2=Enforced.

Once the examined instance is modified, the changes must be committed back to the system. This update is executed at lines 293 through 296. Once completed, the script displays a message showing the current configuration.

To configure a quota per user, as mentioned previously, the *Win32_DiskQuota* with its associations must be used. This is the purpose of the next sample, also written in Jscript. Its command-line parameters are as follows:

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

Usage: WMIQuota.wsf /Action:value /Account:value /Volume:value [/Limit:value]
           [/WarningLimit:value] [/Machine:value] [/User:value] [/Password:value]
```

Options:

Action	:	Specify the operation to perform: [List], [Create], [Update] and [Delete].
Account	:	Set the account associated with the quota.
Volume	:	Set the volume associated with the quota.
Limit	:	Set the limit for the quota in MB.
WarningLimit	:	Set the warning limit for the quota in MB.
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)

Examples:

```
WMIQuota.wsf /Action>List
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Limit:512 /WarningLimit:256
               /Action>Create
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Limit>NoLimit /WarningLimit:256
               /Action>Create
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Limit:512 /WarningLimit>NoLimit
               /Action>Create
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Limit:1024 /Action:Update
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Limit>NoLimit /Action:Update
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /WarningLimit:512 /Action:Update
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /WarningLimit>NoLimit
               /Action:Update
WMIQuota.wsf /Account:LISSWARENET\Alain.Lissoir /Volume:C: /Action>Delete
```

Because the command-line parameters and parsing always use the same structure, Sample 3.36 does not show this portion of the code. We also skipped the portion of the code showing all available instances with their properties, since it also uses a script logic used many times now (lines 203 through 316). The first interesting piece of code in Sample 3.36 concerns the user quota creation (lines 319 through 389).

→ **Sample 3.36 Viewing, creating, updating, and deleting volume quota per user (Part I)**

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
35:   </runtime>
36:
```



```
352:     objWMIInstance.User = objWMIUUserInstance.Path_.RelPath;
353:     objWMIInstance.QuotaVolume = objWMIDiskInstance.Path_.RelPath;
354:
355:     if (varLimit == null)
356:     {
357:         varLimit = cQuotaNoLimit;
358:     }
359:
360:     if (varWarningLimit == null)
361:     {
362:         varWarningLimit = cQuotaNoLimit;
363:     }
364:
365:     objWMIInstance.Limit = varLimit;
366:     objWMIInstance.WarningLimit = varWarningLimit;
367:
368:     try
369:     {
370:         objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate | wbemFlagReturnWhenComplete);
371:     }
372:
373:     if (objWMIInstance.Limit == cQuotaNoLimit)
374:     {
375:         varLimit = "No Limit";
376:     }
377:     if (objWMIInstance.WarningLimit == cQuotaNoLimit)
378:     {
379:         varWarningLimit = "No Limit";
380:     }
381:
382:     WScript.Echo ("Quota on '" + strDeviceID + "' for '" +
383:                   strDomain + "\\" + strName + "' created (" +
384:                   varWarningLimit + " (bytes) / " + varLimit + " (bytes)).");
385:
386: }
387:
388:
389:
```

Because the *Win32_DiskQuota* class is an association class, its creation is a bit unusual, because it is made up of references. First, a new instance of the *Win32_DiskQuota* is created (lines 321 through 330) and because the class is an association class that associates a *Win32_UserAccount* and a *Win32_LogicalDisk*, the code must retrieve these associated instances by using the information given on the command-line parameters (*/Account* and */Volume* switches). This operation is executed from line 332 through 336 for the *Win32_Account* instance and from line 342 through 346 for the *Win32_LogicalDisk* instance. Once these two instances are available, their WMI paths are assigned to the *Win32_DiskQuota* references (lines 352 and 353). If there is no quota limit specified for the user quota creation, by default the script does not define a limit (lines 355 through 363). Next, the script commits the changes to the system (lines 368 through 371).

Besides the user quota creation, the script is also able to handle existing user quota modifications. This portion of the code is shown in Sample 3.37.

Sample 3.37 *Updating volume quota per user (Part II)*

```
...:  
...:  
...:  
390:  
391: // -- UPDATE -----  
392: if (boolUpdate == true)  
393: {  
394:     try  
395:     {  
396:         objWMIInstance = objWMIServices.Get ("Win32_DiskQuota.QuotaVolume="" +  
397:                                         "Win32_LogicalDisk.DeviceID="" +  
398:                                         strDeviceID + "'\" +  
399:                                         ",User="" + "Win32_Account.Domain="" +  
400:                                         strDomain + "',Name="" + strName + "'\"");  
401:     }  
...:  
407:     if (varLimit == null)  
408:     {  
409:         varLimit = objWMIInstance.Limit;  
410:     }  
411:     else  
412:     {  
413:         objWMIInstance.Limit = varLimit;  
414:     }  
415:  
416:     if (varWarningLimit == null)  
417:     {  
418:         varWarningLimit = objWMIInstance.WarningLimit;  
419:     }  
420:     else  
421:     {  
422:         objWMIInstance.WarningLimit = varWarningLimit;  
423:     }  
424:  
425:     try  
426:     {  
427:         objWMIInstance.Put_ (wbemChangeFlagUpdateOnly | wbemFlagReturnWhenComplete)  
428:     }  
...:  
434:     if (objWMIInstance.Limit == cQuotaNoLimit)  
435:     {  
436:         varLimit = "No Limit";  
437:     }  
438:     if (objWMIInstance.WarningLimit == cQuotaNoLimit)  
439:     {  
440:         varWarningLimit = "No Limit";  
441:     }  
442:  
443:     WScript.Echo ("Quota on '" + strDeviceID + "' for '" +  
444:                   strDomain + "\\" + strName + "' updated (" +  
445:                   varWarningLimit + " (bytes) / " + varLimit + " (bytes)).");
```

```
446:     }
447:
...:
...:
...:
```

To update a user quota, we must first retrieve the user quota instance. This is done from line 396 through 400. The WMI path of a *Win32_DiskQuota* instance is made up of the Key properties. This is why it is a bit more complex to code. For example, a *Win32_DiskQuota* path will be coded as follows:

```
Win32_DiskQuota.QuotaVolume="Win32_LogicalDisk.DeviceID=\\"C:\\\"",
    User="Win32_Account.Domain=\\"NET-DPEN6400A\\",
    Name=\\"Administrators\\""
```

Once the *Win32_DiskQuota* instance is retrieved, the miscellaneous settings will be configured based on the command-line parameters given (lines 407 through 423) and committed back to the system (lines 425 through 428). The logic used here is exactly the same as in Sample 3.35—it takes care of the parameters not specified on the command line.

The last supported operation is the deletion of a user quota (see Sample 3.38). This operation has nothing unusual about it. It retrieves the user quota instance to delete (lines 453 through 457), and then deletes that instance with the *Delete_* method of the **SWBemObject** (lines 464 through 467).

→ **Sample 3.38** *Deleting volume quota per user (Part III)*

```
...:
...:
...:
447:
448: // -- DELETE -----
449: if (boolDelete == true)
450: {
451:     try
452:     {
453:         objWMIInstance = objWMIServices.Get ("Win32_DiskQuota.QuotaVolume=\"" +
454:                                         "Win32_LogicalDisk.DeviceID=\"" +
455:                                         strDeviceID + "\\" +
456:                                         ",User=\"" + "Win32_Account.Domain=\"" +
457:                                         strDomain + "\",Name=\"" + strName + "\\"");
458:     }
459:
460:     try
461:     {
462:         objWMIInstance.Delete_();
463:     }
464:
465:     WScript.Echo ("Quota on '" + strDeviceID + "' for '" +
466:                   strDomain + "\\" + strName + "' deleted.");
467:
468:
469:
470:
471:
472:
473:
474:
```

```

475:      }
476:
477:  ]]>
478:  </script>
479: </job>
480:</package>

```

3.5.2 DFS provider

Distributed File System (DFS) is the ability to logically group shares from multiple servers and to link these shares transparently. All the linked shares appear in a treelike structure within a single Root. The *DFS* provider supports the configuration and the management of DFS in Windows Server 2003. Table 3.44 shows the *DFS* provider capabilities.

Table 3.44

The DFS Providers 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
DFS Provider	Root\CIMv2	X	X			X	X	X	X							
DFSPublisher	Root\DFSPublisher															

Table 3.45 summarizes the classes supported by the *DFS* provider. All classes are available in the Root\CIMv2 namespace.

Table 3.45

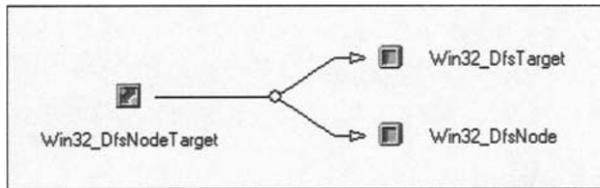
The DFS Provider Classes

Name	Type	Comments
Win32_DfsTarget	Dynamic	The DfsTarget class represents a target of a DFS link.
Win32_DfsNode	Dynamic	The Win32_DfsNode class represents a root or a link of a domain based or a standalone distributed file system (DFS).
Win32_DfsNodeTarget	Association	The Win32_DfsNodeTarget class associates a DFS node to one of its targets.

With this provider it is possible to create DFS nodes and use associations to represent the connections between nodes, shares, and servers linked to the nodes. Figure 3.22 represents this association.

A DFS Root can be defined at the domain level for domain-based operation or at the server level for standalone operation. Domain-based DFS can have multiple Roots in the domain but only one Root on each server. The *Create* method of the *Win32_DfsNode* can be used to create new nodes.

Figure 3.22
The Win32_DFSNodeTarget association class.



Sample 3.39 and 3.40 show how to retrieve, create, update, and delete DFS nodes. The script exposes the following command-line parameters:

```

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

Usage: WMIDfs.wsf /Action:value /Share:value /Server:value /RootDFS:value
       [/LinkState:value] [/TimeOut:value] [/Description:value]
       [/Machine:value] [/User:value] [/Password:value]

Options:

Action      : Specify the operation to perform: [ViewRootNodes], [ViewNodes],
              [Create], [Update] and [Delete].
Share       : The ShareName indicates the name of the share that the link references.
Server      : The ServerName indicates the name of the server that the link references.
RootDFS    : The Root DFS Node that represents a link of a domain based or a standalone DFS.
LinkState   : Indicates the state of the DFS link. Only [Offline], [Online] or
              [Active] are accepted.
TimeOut     : Indicates the time in seconds for which the client caches the referral of a node.
Description : Textual description of a Node.
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)

Examples:
  
```

```

WMIDFS.Wsf /Action:ViewRootNodes
WMIDFS.Wsf /Action:ViewNodes
WMIDFS.Wsf /Action>Create /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot /Description:"DFS Node to SubDirectory_1"
WMIDFS.Wsf /Action:Update /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot /Timeout:10
WMIDFS.Wsf /Action:Update /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot /LinkState:OffLine
WMIDFS.Wsf /Action:Update /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot /LinkState:OnLine
WMIDFS.Wsf /Action:Update /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot /LinkState:Active
WMIDFS.Wsf /Action:Delete /Share:SubDirectory_9 /Server:NET-DPEN6400A.LissWare.Net
            /RootDFS:\\LISSWARENET\MyDFSRoot
  
```

The first part of the script (Sample 3.39) defines and parses the command-line parameters (skipped lines 13 through 36 and lines 86 through 160). Next, after the WMI connection (lines 162 through 165), based on the value of the **/Action** switch, the script retrieves the *Win32_DFSNode* instances (lines 169 through 239).

→ **Sample 3.39** Viewing, creating, modifying, and deleting DFS nodes (Part I)

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
36:   </runtime>
37:
38:   <script language="VBScript" src=..\Functions\DecodeDFSStateFunction.vbs*>
39:
40:   <script language="VBScript" src=..\Functions\DisplayFormattedPropertyFunction.vbs*>
41:   <script language="VBScript" src=..\Functions\TinyErrorHandler.vbs*>
42:
43:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
44:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
45:
46:   <script language="VBScript">
47:     <![CDATA[
.:
51:     Const cComputerName = "LocalHost"
52:     Const cWMINameSpace = "Root/cimv2"
53:     Const cWMIDfsNodeClass = "Win32_DfsNode"
54:     Const cWMIDfsTargetClass = "Win32_DfsTarget"
.:
84:   ' -----
85:   ' Parse the command line parameters
86:   If WScript.Arguments.Named.Count = 0 Then
87:     WScript.Arguments.ShowUsage()
88:     WScript.Quit
89:   End If
90:
91:   Select Case Ucase(WScript.Arguments.Named("Action"))
92:     Case "VIEWROOTNODES"
93:       boolList = True
94:       boolDFSRootNodes = True
95:     Case "VIEWNODES"
96:       boolList = True
97:       boolDFSRootNodes = False
98:     Case "CREATE"
99:       boolCreate = True
100:    Case "UPDATE"
101:      boolUpdate = True
102:    Case "DELETE"
103:      boolDelete = True
104:    Case Else
105:      WScript.Echo "Invalid action type. Only [List], [Create], [Update] ...
106:      WScript.Arguments.ShowUsage()
107:      WScript.Quit
108:   End Select
.:
159:   strComputerName = WScript.Arguments.Named("Machine")
160:   If Len(strComputerName) = 0 Then strComputerName = cComputerName
161:
162:   objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
163:   objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
```

```
164:
165:     Set objWMIServices = objWMIConnector.ConnectServer(strComputerName, cWMINamespace, _
166:                                         strUserID, strPassword)
...
169:     ' -- LIST -----
170:     If boolList = True Then
171:
172:         Set objWMIInstances = objWMIServices.InstancesOf (cWMIDfsNodeClass)
...
175:     If objWMIInstances.Count Then
176:         For Each objWMIInstance In objWMIInstances
177:             If boolDFSRootNodes = objWMIInstance.Root Then
178:                 WScript.Echo "- Node: " & objWMIInstance.Name & String (60, "-")
179:
180:                 Set objWMIPropertySet = objWMIInstance.Properties_
181:                 For Each objWMIProperty In objWMIPropertySet
182:                     Select Case objWMIProperty.Name
183:                         Case "Caption"
184:
185:                             Case "State"
186:                                 DisplayFormattedProperty objWMIInstance, _
187:                                         " " & objWMIProperty.Name, _
188:                                         DecodeDFSNodeState (objWMIProperty.Value), _
189:                                         Null
190:                         Case Else
191:                             DisplayFormattedProperty objWMIInstance, _
192:                                         " " & objWMIProperty.Name, _
193:                                         objWMIProperty.Value, _
194:                                         Null
195:                     End Select
196:                 Next
...
199:                 Set objWMIAssocInstances = objWMIServices.ExecQuery -
200:                                         ("Associators of {" & _
201:                                         objWMIInstance.Path_.RelPath & "}")
202:
203:                 For Each objWMIAssocInstance In objWMIAssocInstances
204:                     WScript.Echo vbCrLf & " - Target Link: \\" & _
205:                                         objWMIAssocInstance.ServerName & _
206:                                         "\\" & objWMIAssocInstance.ShareName & _
207:                                         " " & String (60, "-")
208:                     Set objWMIPropertySet = objWMIAssocInstance.Properties_
209:                     For Each objWMIProperty In objWMIPropertySet
210:                         Select Case objWMIProperty.Name
211:                             Case "Caption"
212:
213:                             Case "State"
214:                                 DisplayFormattedProperty objWMIInstance, _
215:                                         " " & objWMIProperty.Name, _
216:                                         DecodeDFSTargetState (objWMIProperty.Value), _
217:                                         Null
218:                         Case Else
219:                             DisplayFormattedProperty objWMIInstance, _
220:                                         " " & objWMIProperty.Name, _
221:                                         objWMIProperty.Value, _
222:                                         Null
223:                     End Select
224:                 Next
...
226:             Next
```

```

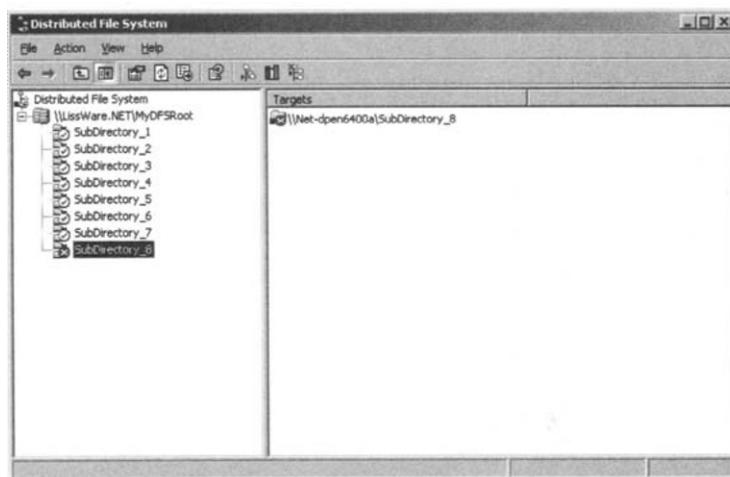
...:
230:           WScript.Echo
231:           End If
232:       Next
233:   Else
234:       WScript.Echo "No DFS nodes available."
235:   End If
...:
239: End If
240:
...:
...:
...

```

At line 172, the script retrieves all instances from the *Win32_DFSNode* class. Next, it uses the usual scripting technique to display the *Win32_DFSNode* instance properties (lines 176 through 232). The **/Action:ViewRootNodes** or **/Action:ViewNodes** switches determine if the DFS node Root instances must be displayed by performing a comparison of the switch value with the *Root* property of the *Win32_DFSNode* instance (line 177). Next, it displays the properties of the instance accordingly (lines 180 through 196).

As shown in Figure 3.22, the *Win32_DFSNode* class is associated with the *Win32_DFSTarget* by the *Win32_DfsNodeTarget* association class. The script takes advantage of this association (lines 199 through 201) to display information about the *Win32_DFSTarget* instances (lines 203 through 224).

Figure 3.23
A DFS
configuration
example.



For a configuration similar to the one shown in Figure 3.23, the script execution with the **/Action:ViewRootNodes** switch will produce an output similar to the following one:

```
1: C:\>WMIDFS.Wsf /Action:ViewRootNodes
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Node: \\LISSWARENET\MyDFSRoot-----
6:   Description: ..... My Domain DFS Root
7:   Name: ..... \\LISSWARENET\MyDFSRoot
8:   Root: ..... True
9:   State: ..... Ok
10:  Timeout: ..... 30
11:
12: - Target Link: \\NET-DPEN6400A\MyDFSRoot -----
13:  LinkName: ..... \\LISSWARENET\MyDFSRoot
14:  ServerName: ..... NET-DPEN6400A
15:  ShareName: ..... MyDFSRoot
16:  State: ..... OnLine.
```

From line 5 through 10, we see the *Win32_DFSNode* instance properties and from line 12 through 16, we see the *Win32_DFSTarget* instance properties associated with it.

The script execution with the **/Action:ViewNodes** switch will produce an output similar to the following one:

```
1: C:\>WMIDFS.Wsf /Action:ViewNodes
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Node: \\LISSWARENET\MyDFSRoot\SubDirectory_3-----
6:   Description: ..... SubDirectory_3
7:   Name: ..... \\LISSWARENET\MyDFSRoot\SubDirectory_3
8:   Root: ..... False
9:   State: ..... OnLine
10:  Timeout: ..... 10
11:
12: - Target Link: \\NET-DPEN6400A.LissWare.Net\SubDirectory_3 -----
13:  LinkName: ..... \\LISSWARENET\MyDFSRoot\SubDirectory_3
14:  ServerName: ..... NET-DPEN6400A.LissWare.Net
15:  ShareName: ..... SubDirectory_3
16:  State: ..... OnLine.
17:
18: - Node: \\LISSWARENET\MyDFSRoot\SubDirectory_8-----
19:   Description: ..... SubDirectory_8
20:   Name: ..... \\LISSWARENET\MyDFSRoot\SubDirectory_8
21:   Root: ..... False
22:   State: ..... OnLine
23:   Timeout: ..... 10
24:
25: - Target Link: \\NET-DPEN6400A.LissWare.Net\SubDirectory_8 -----
26:  LinkName: ..... \\LISSWARENET\MyDFSRoot\SubDirectory_8
27:  ServerName: ..... NET-DPEN6400A.LissWare.Net
28:  ShareName: ..... SubDirectory_8
29:  State: ..... OffLine.
...:
...:
...:
```

The output is similar to the previous one. However, only *Win32_DFS-Node* non-Root instances are displayed. You will notice that the state of the “SubDirectory_8” is off line (line 29), as shown in Figure 3.23.

Sample 3.40, which is the second part of the script, creates (lines 242 through 259), modifies (lines 262 through 290), and deletes (lines 293 through 304) existing *Win32_DFSNode* instances.

Sample 3.40

Viewing, creating, modifying, and deleting DFS nodes (Part II)

```
...:
...:
...:
240:
241: ' -- CREATE -----
242: If boolCreate = True Then
243:     Set objWMIClass = objWMIServices.Get (cWMIDfsNodeClass)
...:
246:     intRC = objWMIClass.Create (strRootDFS & "\" & strShare, _
247:                                     strServer, _
248:                                     strShare, _
249:                                     strDescription)
250:
251: If intRC = 0 Then
252:     WScript.Echo "DFS node '" & strRootDFS & "\" & strShare & "' successfully created."
253: Else
254:     WScript.Echo "Failed to create DFS node '" & strRootDFS & "\" & _
255:                           strShare & "' (0x" & Hex (IntRC) & ")."
256: End If
...:
259: End If
260:
261: ' -- UPDATE -----
262: If boolUpdate = True Then
263:     if intTimeout Then
264:         Set objWMIInstance = objWMIServices.Get (cWMIDfsNodeClass & "=" & _
265:                                         strRootDFS & "\" & strShare & "'")
...:
268:     objWMIInstance.Timeout = intTimeout
269:
270:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete)
...:
272: End If
273:
274: If intLinkState Then
275:     Set objWMIInstance = objWMIServices.Get (cWMIDfsTargetClass & ".LinkName=''" & _
276:                                         strRootDFS & "\" & strShare & _
277:                                         "'", ServerName=''" & strServer & _
278:                                         "'", ShareName=''" & strShare & "'")
...:
281:     objWMIInstance.State = intLinkState - 1
282:
283:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete)
...:
285: End If
286:
287: WScript.Echo "DFS Node '" & strRootDFS & "\" & strShare & "' successfully updated."
```

```
...:  
290: End If  
291:  
292: ' -- DELETE -----  
293: If boolDelete = True Then  
294:     Set objWMIInstance = objWMIServices.Get (cWMIDfsNodeClass & "=" & _  
295:                                         strRootDFS & "\\" & strShare & "")  
...:  
298:     objWMIInstance.Delete_  
...:  
301:     WScript.Echo "DFS Node '" & strRootDFS & "\\" & strShare & "' successfully deleted."  
...:  
304: End If  
...:  
308: []>  
309: </script>  
310: </job>  
311:</package>
```

The creation of a *Win32_DFSNode* instance is performed with the *Create* static method exposed by the *Win32_DFSNode* class. This method requires four parameters, as follows:

- The DFS path parameter, which specifies the path of the DFS Root (which comes from the command line with the /RootDFS switch).
- The Server *name* parameter, which specifies the name of the server that hosts the share to which the DFS link is associated (which comes from the command line with the /Server switch).
- The *ShareName* parameter, which specifies the name of the share to which the DFS link is associated (which comes from the command line with the /Share switch).
- The *Description* parameter, which specifies a comment describing the DFS node (which comes from the command line with the /Description switch).

The *Win32_DFSNode* instance creation is executed at line 246. Next, the script tests if the *Create* method returned a value different from zero. Any value returned from the execution of this method states that an error occurred. These values correspond to the Win32 errors (where corresponding messages are available with a “Net HelpMsg <value>” command).

The *Win32_DFSNode* class and the *Win32_DFSTarget* class allow the modification of some properties of the instances they represent. The *Win32_DFSNode* exposes a *Timeout* property to indicate the time in seconds for which the client caches the referral of this node. The script can modify this property (lines 264 through 270) by retrieving the corresponding *Win32_DFSNode* instance in an *SWBemObject* object (lines 264 and 265) and invoking the *Put_* method (line 270).

In the same way, it is possible to modify the state of a *Win32_DFSTarget* instance by changing the value of the *State* property (lines 274 through 285). The *State* property indicates the state of the DFS target. Note that both *Win32_DFSNode* and *Win32_DFSTarget* classes expose a *State* property. Table 3.46 shows the meaning of the different values for each class.

Table 3.46

The State Property Meaning of the Win32_DFSNode and Win32_Target Classes

<i>Win32_DFSNode State</i> property values	
Meaning	Values
Ok	0
Inconsistent	1
OnLine	2
OffLine	3

<i>Win32_DFSTarget State</i> property values	
Meaning	Values
Offline	0
Online	1
Active	4

To delete a *Win32_DFSNode* instance, the script retrieves an instance of a DFS node (lines 294 and 295) and invokes the *Delete_* method of the *SWBemObject* representing the *Win32_DFSNode* instance (line 298).

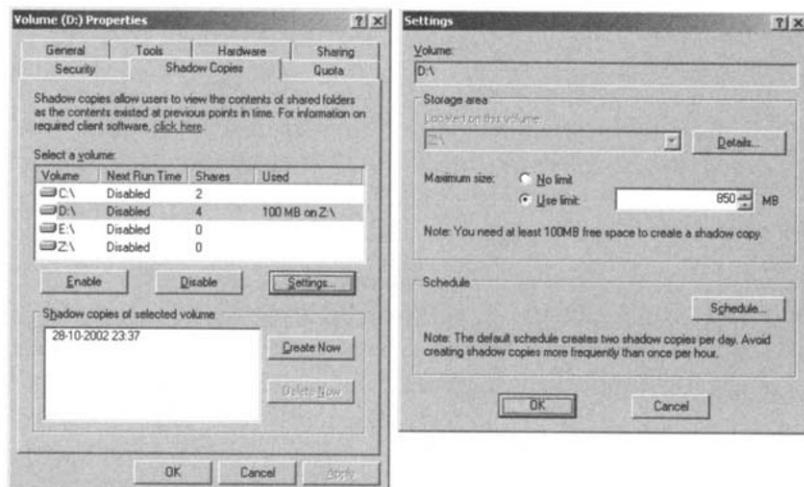
3.5.3

Shadow Copy providers

The *Shadow Copy* providers provide management capabilities to the Windows Server 2003 shadow copy services. The user interface exposing information related to this new feature is available by right-clicking on a volume,

Figure 3.24

Managing the Shadow Copies of a volume from the user interface.



selecting properties, and clicking again on the Shadow Copies pane (see Figure 3.24). Bear in mind that this feature is only available under Windows Server 2003.

The set of properties and actions available from the user interface is also accessible from WMI. The *Shadow Copy* providers supporting these features are registered in the Root\CLMv2 namespace of the CIM repository (see Table 3.47).

→ **Table 3.47** *The Shadow Copy Providers 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
Shadow Copy Providers																
MSVSS PROVIDER	Root\CLMv2	X	X					X	X	X	X	X				
MSVDS PROVIDER	Root\CLMv2	X	X					X	X	X	X	X				

These providers support a set of classes exposing properties and methods to perform most of the Shadow Copy tasks (see Table 3.48). The most relevant classes to use from a scripting point of view are the *Win32_Volume*, *Win32_ShadowCopy*, and *Win32_ShadowStorage* classes.

→ **Table 3.48** *The Shadow Copy Providers Classes*

Name	Type	Comments
Win32_ShadowCopy	Dynamic	The Win32_ShadowCopy class is a storage extent that represents a duplicate copy of the original volume at some previous time.
Win32_ShadowProvider	Dynamic	The Win32_ShadowProvider class represents a component, typically a combination of user-mode and kernel/firmware implementation, that will perform the work involved in creating and representing volume shadow copies.
Win32_Volume	Dynamic	The Win32_Volume class represents an area of storage on a hard disk. The class returns local volumes that are formatted, unformatted, mounted, or offline. A volume is formatted by using a file system, such as FAT or NTFS, and may have a drive letter assigned to it. A single hard disk can have multiple volumes, and volumes can also span multiple disks.
Win32_MountPoint	Association	The mount point associates a volume to the directory at which it is mounted.
Win32_ShadowBy	Association	The association between a shadow copy and the provider that created the shadow copy.
Win32_ShadowDiffVolumeSupport	Association	The association between a shadow copy provider and a volume supported for differential storage area.
Win32_ShadowFor	Association	The association between a shadow copy and the volume for which the shadow was created.
Win32_ShadowOn	Association	The association between a shadow copy and the volume on which differential data is written.
Win32_ShadowStorage	Association	The association between the volume for which a shadow copy is made and the volume to which the differential data is written.
Win32_ShadowVolumeSupport	Association	The association between a shadow copy provider and a supported volume.
Win32_VolumeQuota	Association	The Win32_VolumeQuota association relates a volume to the per volume quota settings.
Win32_VolumeUserQuota	Association	The Win32_VolumeUserQuota association relates per user quotas to quota-enabled volumes. System administrators can configure Windows to prevent further disk space use and log an event when a user exceeds a specified disk space limit. They can also log an event when a user exceeds a specified disk space warning level. Note that disk quotas cannot be set for the Administrator accounts themselves.

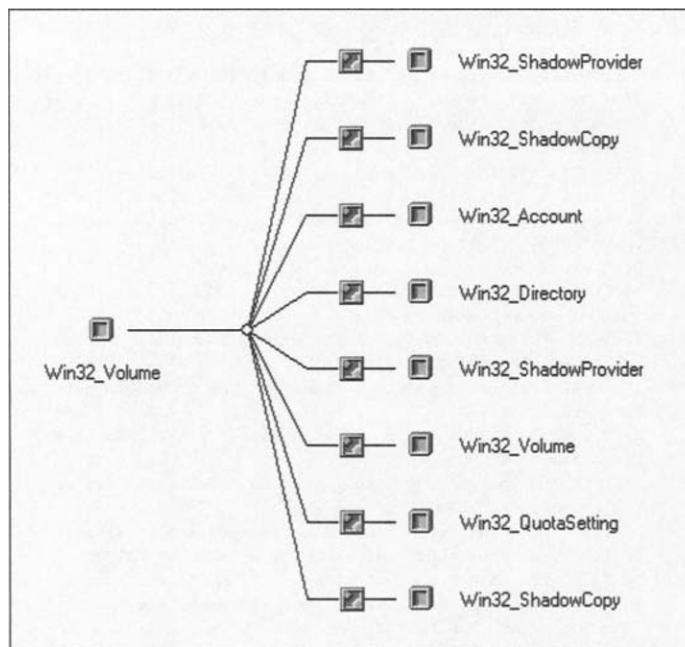
It is interesting to note that the *Win32_Volume* class is quite similar to the *Win32_LogicalDisk* class. For instance, it is possible to invoke the *Chkdsk* method from the *Win32_Volume* class, as we did in Sample 2.3 with the *Win32_LogicalDisk* class. Both *Chkdsk* method implementations expose the exact same input parameters. However, the *Win32_Volume* class exposes some extra properties and methods especially related to the Shadow Copy features. Actually, during the design phase of the Shadow Copy object model, it had been noted that the current *Win32_LogicalDisk* class was not providing enough information to suit the requirements. Therefore, Microsoft decided to create the *Win32_Volume* class to suit its needs. That's why this class implements some methods of the *Win32_LogicalDisk* class (i.e., *Chkdsk*, *ExcludeFromAutoChk*, *ScheduleAutoChk*) with some new methods and functionalities to request information about the defragmentation analysis, perform a defragmentation, format a volume (i.e., *Defrag*, *DefragAnalysis*, *Format*), mount or dismount a volume, and manage shadow copies. However, there are several noteworthy differences between *Win32_LogicalDisk* and *Win32_Volume* classes. These are as follows:

- The *Win32_Volume* class does not manage floppy disk drives.
- The *Win32_Volume* class can be used to change the volume drive letter, while the *Win32_LogicalDisk* does not support this.
- *Win32_Volume* class enumerates all volumes, not just those with drive letters similar to the *Win32_LogicalDisk* class.
- The *Win32_Volume* class does not enumerate network shares that are mapped to drive letters similar to the *Win32_LogicalDisk* class.

Besides the *Win32_Volume* class, another very interesting class is the *Win32_ShadowCopy* class. Instances of this class represent the shadow copies of the original volume at some previous time. The *Win32_Volume* and *Win32_ShadowCopy* classes are associated with the *Win32_ShadowFor* and *Win32_ShadowOn* association classes (see Figure 3.25). The *Win32_ShadowCopy* class exposes the *Create* method to create shadow copies, which corresponds to the “Create Now” button shown in Figure 3.24.

The final, most relevant class is the *Win32_ShadowStorage* class. This class is an association class linking the *Win32_Volume* class with the *Win32_Volume* class itself. The purpose of this association class is to expose some properties visible in the user interface (see Figure 3.24—i.e., used space, allocated space, and the maximum space) and create an association between a volume that contains information to be shadowed and a volume containing the shadows. Some interesting properties of the *Win32_ShadowStorage* class are the *ShadowedVolume* and *ShadowVolume* properties, which are both of type *Win32_Volume*.

Figure 3.25
The Win32_Volume class and its associations.



owStorage class are, for example, *AllocatedSpace*, *MaxSpace*, and *UsedSpace*, where *MaxSpace* can be updated.

A very interesting aspect of the Shadow Copy object model is the existence of associations between the *Win32_Volume* class and the *Win32_QuotaSetting* class (see Figure 3.25). Similar to the *Win32_LogicalDisk* class, by managing the volumes with the *Win32_Volume* class, it is possible to retrieve and manage disk quota information. However, we will not cover the disk quota management in the next script sample, since the logic and coding technique are exactly the same as samples working with the *Win32_LogicalDisk* and *Win32_QuotaSettings* classes (see Samples 3.33 through 3.38).

Samples 3.41 through 3.52 illustrate how to work with these three Shadow Copy classes. The command-line parameters supported by the script expose most methods and manageable properties of the classes.

```

C:\>WMIDiskSvc.wsf /?
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Usage: WMIDiskSvc.wsf Volume [/Action:value] [/ShadowCopy[+|-]] [/Force[+|-]] [/FileSystem:value]
           [/ClusterSize:value] [/Label:value] [/Compression[+|-]]
           [/FixErrors[+|-]] [/VigorousIndexCheck[+|-]] [/SkipFolderCycle[+|-]]
           [/ForceDismount[+|-]] [/RecoverBadSectors[+|-]] [/OKToRunAtBootUp[+|-]]
           [/MaxSpace:value] [/ShadowStorage:value] [/ShadowCopyID:value]
           [/Machine:value] [/User:value] [/Password:value]
  
```

Options:

Volume	: The logical disk letter (i.e., C: or D:).
Action	: Determines the action to perform. Only [List], [Format], [DefragAnalysis], [Defrag], [Dismount], [Chkdsk], [CreateShadowCopy], [DeleteShadowCopy], [CreateShadowStorage], [UpdateShadowStorage] or [DeleteShadowStorage] is accepted.
ShadowCopy	: Only list the shadow copies available on the system.
Force	: Forces the defrag even if free space on the disk is low.
FileSystem	: Determines the filesystem to use for the formatted volume. Only [NTFS], [FAT] or [FAT32].
ClusterSize	: Determines the volume cluster size to use.
Label	: Determines the volume label.
Compression	: Enables the compression on the formatted volume.
FixErrors	: Indicates what should be done to errors found on the disk. If true, then errors are fixed. The default is FALSE.
VigorousIndexCheck	: If TRUE, a vigorous check of index entries should be performed. The default is TRUE.
SkipFolderCycle	: If TRUE, the folder cycle checking should be skipped or not. The default is TRUE.
ForceDismount	: If TRUE, the drive should be forced to dismount before checking. The default is FALSE.
RecoverBadSectors	: If TRUE, the bad sectors should be located and the readable information should be recovered from these sectors. The default is FALSE.
OKToRunAtBootUp	: If TRUE, the chkdsk operation should be performed at next boot up time, in case the operation could not be performed because the disk was locked at time the method was called. The default is FALSE.
MaxSpace	: Determines the maximum space to be used on the Shadow Storage.
ShadowStorage	: Determines the volume to use to store the Shadows.
ShadowCopyID	: Specifies the shadow copy ID to delete.
Machine	: Determines the WMI system to connect to. (default=LocalHost)
User	: Determines the UserID to perform the remote connection. (default=None)
Password	: Determines the password to perform the remote connection. (default=None)

Example:

```

WMIDiskSvc.wsf /Action>List
WMIDiskSvc.wsf /Action>List /ShadowCopy+

WMIDiskSvc.wsf C: /Action:DefragAnalysis
WMIDiskSvc.wsf C: /Action:Defrag /Force+

WMIDiskSvc.wsf C: /Action:Format /FileSystem:NTFS
    /QuickFormat+ /ClusterSize:4096 /Label:"MyDisk" /Compression+

WMIDiskSvc.wsf C: /Action:Chkdsk /FixErrors+ /VigorousIndexCheck+ /SkipFolderCycle+
WMIDiskSvc.wsf C: /Action:Chkdsk /ForceDismount+ /RecoverBadSectors+ /OKToRunAtBootUp+

WMIDiskSvc.wsf C: /Action>CreateShadowStorage /MaxSpace:850 /ShadowStorage:Z:
WMIDiskSvc.wsf C: /Action:UpdateShadowStorage /MaxSpace:Unlimited
WMIDiskSvc.wsf C: /Action>DeleteShadowStorage

WMIDiskSvc.wsf C: /Action>CreateShadowCopy
WMIDiskSvc.wsf /Action>DeleteShadowCopy /ShadowCopyID:{a2e1d5d6-4bab-4bd8-b133-dad5798ec9ff}

```

As usual, the first part of the script defines (skipped lines 13 through 59) and parses (skipped lines 130 through 274) the command-line parameters. Next, it executes the WMI connection (lines 276 through 280).

→ **Sample 3.41** Managing disk services and shadow copies (Part I)

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
59:  </runtime>
60:
61:  <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertiesFunction.vbs" />
62:  <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
63:  <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
64:
65:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
66:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
67:
68:  <script language="VBScript">
69:  <![CDATA[
.:
73:  Const cComputerName = "LocalHost"
74:  Const cWMINameSpace = "root/cimv2"
75:
76:  Const cUnlimitedMaxSpace = "18446744073709551615"
.:
130:  ' -----
131:  ' Parse the command line parameters
132:  strAction = WScript.Arguments.Named("Action")
133:  Select Case UCase (strAction)
134:      Case "LIST"
135:          boolShadowCopy = WScript.Arguments.Named("ShadowCopy")
136:          If Len(boolShadowCopy) = 0 Then boolShadowCopy = False
137:
138:          boolList = True
139:
140:      Case "CHKDSK"
.:
267:  strUserID = WScript.Arguments.Named("User")
268:  If Len(strUserID) = 0 Then strUserID = ""
269:
270:  strPassword = WScript.Arguments.Named("Password")
271:  If Len(strPassword) = 0 Then strPassword = ""
272:
273:  strComputerName = WScript.Arguments.Named("Machine")
274:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
275:
276:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
277:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
278:
279:  Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
280:                                              strUserID, strPassword)
.:
282:
.:
.:
```

The first feature supported by the script is the ability to list all shadow copies available on a system. This feature requires a very basic scripting technique, since it simply requests all instances of the *Win32_ShadowCopy* class and displays all properties of each instance. This capability will be useful when we will need to delete a shadow copy with WMI. Actually, the key property to retrieve a *Win32_ShadowCopy* instance is a GUID number exposed by the *ID* property. This is why it is interesting to implement a function listing all *Win32_ShadowCopy* instances with their properties (see Sample 3.42).

Sample 3.42

Viewing all Win32_ShadowCopy instances (Part II)

```
...:
...:
...:
282:
283: ' -- List Shadow Copy -----
284: If boolList = True And boolShadowCopy = True Then
285:     Set objWMInstances = objWMIServices.InstancesOf ("Win32_ShadowCopy")
...:
288: If objWMInstances.Count Then
289:     For Each objWMInstance In objWMInstances
290:         objWMIDateTime.Value = objWMInstance.InstallDate
291:
292:         WScript.Echo "- Shadow Copy: (" & objWMIDateTime.GetVarDate (False) & _
293:             ") " & String (60, "-")
294:
295:         Set objWMIPropertySet = objWMInstance.Properties_
296:         For Each objWMIProperty In objWMIPropertySet
297:             DisplayFormattedProperty objWMInstance, _
298:                 objWMIProperty.Name, _
299:                 objWMIProperty.Name, _
300:                 Null
301:             Next
...:
304:         Next
305:     Else
306:         WScript.Echo "No shadow copy available."
307:     End If
308: End If
309:
...:
...:
...:
```

As an example, the script will display the following information:

```
1: C:\>WMIDiskSvc.wsf /Action>List /ShadowCopy+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - Shadow Copy: (28-10-2002 22:37:09) -----
6: ClientAccessible: ..... TRUE
7: Count: ..... 1
8: DeviceObject: ..... \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy13
9: Differential: ..... FALSE
10: ExposedLocally: ..... FALSE
```

```

11: ExposedRemotely: ..... FALSE
12: HardwareAssisted: ..... FALSE
13: *ID: ..... {2f26b83b-d583-461d-b773-e6cd024f9710}
14: Imported: ..... FALSE
15: InstallDate: ..... 28-10-2002 22:37:09
16: NoAutoRelease: ..... TRUE
17: NotSurfaced: ..... FALSE
18: NoWriters: ..... TRUE
19: OriginatingMachine: ..... net-dpep6400a.Emea.LissWare.NET
20: Persistent: ..... TRUE
21: Plex: ..... FALSE
22: ProviderID: ..... {b5946137-7b9f-4925-af80-51abd60b20d5}
23: ReadWrite: ..... FALSE
24: ServiceMachine: ..... net-dpep6400a.Emea.LissWare.NET
25: SetID: ..... {238304a1-3df0-4d0e-9f16-d143abe87a29}
26: State: ..... 12
27: Transportable: ..... FALSE
28: VolumeName: ..... \\?\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\
```

Of course, if we are able to list the shadow copies available in a system, it is possible to create new shadow copies by using the *Win32_ShadowCopy* class and its *Create* method, as shown in Sample 3.43. It is interesting to note that the *Create* method is a static method. Therefore, the method does not relate to a particular *Win32_ShadowCopy* dynamic instance. Instead, an instance of the class must be created to invoke the method (line 314). From line 316 through 318, the method requests two input parameters (the drive letter and the context used by the WMI provider to create the shadow, which is always “Client Accessible”) and returns one output parameter (which is the ID of the created shadow) displayed at line 324.

→ **Sample 3.43** *Creating new shadow copies (Part III)*

```

...:
...:
...:
309:
310: ' -- CreateShadowCopy -----
311: If boolCreateShadowCopy = True Then
312:     WScript.Echo "Creating shadow copy ..."
313:
314:     Set objWMIClass = objWMIServices.Get ("Win32_ShadowCopy")
315:
316:     intRC = objWMIClass.Create (strWMIDriverLetter & "\", _
317:                                 "ClientAccessible", _
318:                                 strShadowCopyID)
...:
321:     If intRC Then
322:         WScript.Echo "Shadow copy creation error (" & intRC & ")."
323:     Else
324:         WScript.Echo "Shadow copy '" & strShadowCopyID & "' successfully created."
325:     End If
326: End If
327:
...:
...:
...:
```

To create a shadow copy, the following command line must be used:

```

1:  C:\>WMIDiskSvc.wsf D: /Action>CreateShadowCopy
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Creating shadow copy ...
6:  Shadow copy '{2f26b83b-d583-461d-b773-e6cd024f9710}' successfully created.

```

To delete shadow copies, the scripting technique is a little bit different from the creation technique. Instead of using a method exposed by the *Win32_ShadowCopy* class, the script (see Sample 3.44) retrieves the *Win32_ShadowCopy* instance to be deleted by referring the ID passed on the command-line parameters with the */ShadowCopyID* switch (lines 332 and 333). Once the instance is retrieved, the script invokes the *Delete_* method exposed by the *SWBemObject* object (line 336) representing the *Win32_ShadowCopy* instance. The following command line will execute the routine shown in Sample 3.44:

```

1:  C:\>WMIDiskSvc.wsf /Action:DeleteShadowCopy /ShadowCopyID:{2f26b83b-d583-461d-b773-e6cd024f9710}
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Deleting shadow copy ...
6:  Shadow Copy '{2f26b83b-d583-461d-b773-e6cd024f9710}' successfully deleted.

```

Sample 3.44 Deleting shadow copies (Part IV)

```

...:
...:
...:
327:
328:  ' -- DeleteShadowCopy -----
329:  If boolDeleteShadowCopy = True Then
330:      WScript.Echo "Deleting shadow copy ..."
331:
332:      Set objWMIInstance = objWMIServices.Get ("Win32_ShadowCopy='"
333:                                         & strShadowCopyID & "'")
...:
336:      objWMIInstance.Delete_
...:
339:      WScript.Echo "Shadow Copy '" & strShadowCopyID & "' successfully deleted."
...:
342:  End If
343:
...:
...:
...:

```

A shadow storage is an association between two volumes (which is simply an association between two *Win32_Volume* instances). To create a shadow storage, the script uses the *Create* static method exposed by the

Win32_ShadowStorage association class (see Sample 3.45, lines 352 through 354).

→ **Sample 3.45** *Associating a shadow storage with a Win32_Volume instance (Part V)*

```
...:
...:
...:
343:
344: ' -- Create Shadow Storage -----
345: If boolCreateShadowStorage = True Then
346:   If Len (strShadowStorage) Then
347:     WScript.Echo "Defining Shadow Storage area for volume '" & _
348:       strWMIDriverLetter & "' on volume '" & _
349:         strShadowStorage & "'..."
350:
351:   Set objWMIClass = objWMIServices.Get ("Win32_ShadowStorage")
352:   intRC = objWMIClass.Create (strWMIDriverLetter & "\", _
353:                             strShadowStorage & "\", _
354:                             varMaxSpace)
355:   If intRC Then
356:     WScript.Echo "Shadow Storage creation error (" & intRC & ")."
357:   Else
358:     WScript.Echo "Shadow Storage '" & strWMIDriverLetter & _
359:       "' successfully defined for volume '" & _
360:         strShadowStorage & "'."
361:   End If
...:
363: End If
364: Else
...:
...:
...:
```

The *Create* method exposes three parameters: the drive letter of the volume containing the data to shadow, the driver letter of the volume used to store the shadows, and the maximum disk space to allocate on that volume. Once the method execution successfully completes, the *Win32_ShadowStorage* association instance is created. The command line to use for this operation is:

```
1: C:\>WMIDiskSvc.wsf D: /Action:CreateShadowStorage /MaxSpace:850 /ShadowStorage:Z:
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Defining Shadow Storage area for volume 'D:' on volume 'Z:'...
6: Shadow Storage 'D:' successfully defined for volume 'Z:'.
```

Samples 3.46 through 3.52 relate to different operations regarding the *Win32_Volume* instances. Sample 3.46 shows how to list all *Win32_Volume* instances with their associated *Win32_ShadowStorage* and *Win32_ShadowCopy* instances. Actually, the script exploits the associations illustrated in Figure 3.25.

Sample 3.46 Viewing the volumes with their related shadow storage and shadow copies (Part VI)

```

...:
...:
...:
364: Else
365:     Set objWMIInstances = objWMIServices.InstancesOf ("Win32_Volume")
...:
368: For Each objWMIInstance In objWMIInstances
369:
370:     ' -- List -----
371:     If boolList = True And boolShadowCopy = False Then
372:         WScript.Echo "- " & objWMIInstance.DriveLetter & " " &
373:             String (60, "-")
374:
375:         Set objWMIPropertySet = objWMIInstance.Properties_
376:         For Each objWMIProperty In objWMIPropertySet
377:             DisplayFormattedProperty objWMIInstance, _
378:                 objWMIProperty.Name, _
379:                 objWMIProperty.Name, _
380:                 Null
381:         Next
...:
384:         Set objWMIAssociatedInstances = objWMIServices.ExecQuery _
385:             ("References of (" & _
386:                 objWMIInstance.Path_.RelPath & _
387:             ") Where Role=Volume " & _
388:             "ResultClass=Win32_ShadowStorage")
...:
391:     For Each objWMIAssociatedInstance In objWMIAssociatedInstances
392:
393:         WScript.Echo vbCRLF & " - Shadow Storage: " & String (60, "-")
394:
395:         Set objWMIPropertySet = objWMIAssociatedInstance.Properties_
396:         For Each objWMIProperty In objWMIPropertySet
397:             DisplayFormattedProperty objWMIAssociatedInstance, _
398:                 " " & objWMIProperty.Name, _
399:                 objWMIProperty.Name, _
400:                 Null
401:         Next
...:
403:     Next
404:
405:     Set objWMIAssociatedInstances = objWMIServices.ExecQuery _
406:         ("Associators of (" & _
407:             objWMIInstance.Path_.RelPath & _
408:         ") Where AssocClass=Win32_ShadowFor")
...:
411:     For Each objWMIAssociatedInstance In objWMIAssociatedInstances
412:
413:         WScript.Echo vbCRLF & " - Shadow Copy: " & _
414:             objWMIAssociatedInstance.ID & _
415:                 " " & String (20, "-")
416:
417:         Set objWMIPropertySet = objWMIAssociatedInstance.Properties_
418:         For Each objWMIProperty In objWMIPropertySet
419:             DisplayFormattedProperty objWMIAssociatedInstance, _
420:                 " " & objWMIProperty.Name, _

```

```
421:                 objWMIProperty.Name, _  
422:                 Null  
423:                 Next  
...:  
425:                 Next  
426:  
427:                 WScript.Echo  
428:             Else  
...:  
...:  
...:
```

First, Sample 3.46 requests all instances of the *Win32_Volume* class (line 365) and performs a loop (Sample 3.46, line 368, through Sample 3.52, line 596) to show the properties of each instance found in the collection (lines 372 through 382). Actually, this loop embraces many more operations than simply showing all instances of the *Win32_Volume* class. It is also used to find a specific instance of the *Win32_Volume* class when one *Win32_Volume* instance must be retrieved to perform a specific operation, such as a check disk or a defragmentation (Sample 3.47, line 429).

But let's come back to Sample 3.46 and how the associated classes are retrieved (lines 384 through 388 and lines 405 through 408). During the loop listing all *Win32_Volume* instances, Sample 3.46 retrieves the associated shadow storage instance (lines 384 through 388). The shadow storage is materialized in the CIM repository by an instance of the *Win32_ShadowStorage* association class. Usually, there is always one *Win32_ShadowStorage* association instance per *Win32_Volume* instance. Therefore, the script uses the following WQL data query to retrieve the *Win32_ShadowStorage* instance (lines 384 through 388):

```
References of {Win32_Volume.DeviceID="\\\\?\\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\\\""}  
Where Role=Volume ResultClass=Win32_ShadowStorage
```

This query is directly inspired by the relationship that exists between the *Win32_Volume* class and the *Win32_ShadowStorage* association class (see Figure 3.25).

In the same way, to retrieve the *Win32_ShadowCopy* instances associated with the *Win32_Volume* instance, the script uses another WQL data query (lines 405 through 408):

```
Associators of {Win32_Volume.DeviceID="\\\\?\\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\\\""}  
Where AssocClass=Win32_ShadowFor
```

Another interesting question, but not implemented in the current script, concerns the volumes using a specific volume for shadow storage. This question can be answered by using the following query:

```
Associators of {Win32_Volume.DeviceID="\\\\?\\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\\\"}
  Where AssocClass=Win32_ShadowStorage ResultRole=Volume"
```

For each instance retrieved by the WQL data queries, the script displays all properties. A sample output would be as follows:

```
1:  C:\>WMIDiskSvc.wsf /Action>List
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  - C: -----
6:  Automount: ..... TRUE
7:  BlockSize: ..... 2048
8:  Capacity: ..... 2144376832
9:  Caption: ..... C:\_
10: Compressed: ..... FALSE
11: *DeviceID: ..... \?\Volume{3eed7422-a3b2-11d6-a5f4-806e6f6e6963}\_
12: DirtyBitSet: ..... FALSE
13: DriveLetter: ..... C:
14: DriveType: ..... 3
15: FileSystem: ..... NTFS
16: FreeSpace: ..... 782168064
17: IndexingEnabled: ..... TRUE
18: InstallDate: ..... 01-01-2000
19: Label: ..... Windows Server 2003
20: MaximumFileNameLength: ..... 255
21: Name: ..... C:\_
22: QuotasEnabled: ..... FALSE
23: QuotasIncomplete: ..... FALSE
24: QuotasRebuilding: ..... FALSE
25: SerialNumber: ..... 615808770
26: SupportsDiskQuotas: ..... TRUE
27: SupportsFileBasedCompression: ..... TRUE
28: SystemName: ..... NET-DPEP6400A
29:
30: - D: -----
31: Automount: ..... TRUE
32: BlockSize: ..... 4096
33: Capacity: ..... 7222730752
...
50: SerialNumber: ..... 821048372
51: SupportsDiskQuotas: ..... TRUE
52: SupportsFileBasedCompression: ..... TRUE
53: SystemName: ..... NET-DPEP6400A
54:
55: - Shadow Storage: -----
56: AllocatedSpace: ..... 104857600
57: *DiffVolume: ..... Win32_Volume.DeviceID=
  "\\\\?\\Volume{3eed7423-a3b2-11d6-a5f4-806e6f6e6963}\\\""
58: MaxSpace: ..... 891289600
59: UsedSpace: ..... 1064960
60: *Volume: ..... Win32_Volume.DeviceID=
  "\\\\?\\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\\\""
61:
62: - Shadow Copy: {2f26b83b-d583-461d-b773-e6cd024f9710} -----
63: ClientAccessible: ..... TRUE
64: Count: ..... 1
65: DeviceObject: ..... \GLOBALROOT\Device\HarddiskVolumeShadowCopy13
66: Differential: ..... FALSE
```

```

67:     ExposedLocally: ..... FALSE
68:     ExposedRemotely: ..... FALSE
69:     HardwareAssisted: ..... FALSE
70:     *ID: ..... {2f26b83b-d583-461d-b773-e6cd024f9710}
71:     Imported: ..... FALSE
72:     InstallDate: ..... 28-10-2002 22:37:09
73:     NoAutoRelease: ..... TRUE
74:     NotSurfaced: ..... FALSE
75:     NoWriters: ..... TRUE
76:     OriginatingMachine: ..... net-dpep6400a.Emea.LissWare.NET
77:     Persistent: ..... TRUE
78:     Plex: ..... FALSE
79:     ProviderID: ..... {b5946137-7b9f-4925-af80-51abd60b20d5}
80:     ReadWrite: ..... FALSE
81:     ServiceMachine: ..... net-dpep6400a.Emea.LissWare.NET
82:     SetID: ..... {238304a1-3df0-4d0e-9f16-d143abe87a29}
83:     State: ..... 12
84:     Transportable: ..... FALSE
85:     VolumeName: ..... \\?\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\
86:
87: - E: -----
88:     Automount: ..... TRUE
89:     BlockSize: ..... 4096
90:     Capacity: ..... 7222759424
91:     Caption: ..... E:\
92:     Compressed: ..... FALSE
93:     *DeviceID: ..... \\?\Volume{3eed7425-a3b2-11d6-a5f4-806e6f6e6963}\

...:
...:
...:
```

Sample 3.47 shows how to use the *Chkdsk* method exposed by the *Win32_Volume* class. This method is not a static method and therefore relates to a specific *Win32_Volume* instance. This is the reason why the method invocation is executed inside the loop (Sample 3.46, line 368, and Sample 3.52, line 596) and for a specific *Win32_Volume* instance (Sample 3.47, line 429). The method parameters and usage are exactly the same as the *Win32_LogicalDisk* method (see Sample 2.3).

Sample 3.47

Executing the Chkdsk Win32_Volume method (Part VII)

```

...:
...:
...:
428:     Else
429:         If Ucase (objWMIInstance.DriveLetter) = Ucase (strWMIDriverLetter) Then
430:
431:             ' -- Chkdsk -----
432:             If boolChkdsk Then
433:                 If boolFixErrors Then
434:                     WScript.Echo "Errors will be fixed."
435:                 End If
436:
437:                 If boolVigorousIndexCheck Then
438:                     WScript.Echo "Vigorous check of index entries will be performed."
```

```
439: End If
440:
441: If boolSkipFolderCycle Then
442:     WScript.Echo "The folder cycle checking will be skipped."
443: End If
444:
445: If boolForceDismount Then
446:     WScript.Echo "The drive will be forced to dismount before checking."
447: End If
448:
449: If boolRecoverBadSectors Then
450:     WScript.Echo "The bad sectors will be ... recovered from these sectors."
451: End If
452:
453: If boolOKToRunAtBootUp Then
454:     WScript.Echo "The chkdsk ... performed at next boot up time."
455: End If
456:
457: WScript.Echo
458: WScript.Echo "Volume " & Ucase (objWMIInstance.DriveLetter) & _
459:             " has " & objWMIInstance.FreeSpace & _
460:             " bytes free on a total of " & _
461:             objWMIInstance.Capacity & " bytes."
462: WScript.Echo "The type of the file system is " & _
463:             objWMIInstance.FileSystem & "."
464: WScript.Echo "Volume is " & objWMIInstance.Label & "."
465: WScript.Echo "Volume Serial Number is " & _
466:             Right ("0000" & Hex(int (objWMIinstance.SerialNumber / 65536)), 4) & _
467:             "-" & -
468:             Right ("0000" & Hex(objWMIinstance.SerialNumber And 65535), 4) & _
469:             "."
470:
471: WScript.Echo "WMI chkdsk started ..."
472: intRC = objWMIInstance.Chkdsk (boolFixErrors, _
473:                                 boolVigorousIndexCheck, _
474:                                 boolSkipFolderCycle, _
475:                                 boolForceDismount, _
476:                                 boolRecoverBadSectors, _
477:                                 boolOKToRunAtBootUp)
...
480: Select Case intRC
481:     Case 0
482:         WScript.Echo "WMI chkdsk completed successfully."
483:     Case 1
484:         WScript.Echo "Locked and chkdsk scheduled on reboot."
485:     Case 2
486:         WScript.Echo "WMI chkdsk failure - Unknown file system."
487:     Case 3
488:         WScript.Echo "WMI chkdsk failure - Unknown error."
489: End Select
490: End If
491:
...

```

A small difference to note is the data type of the volume serial number. The `Win32_LogicalDisk` class exposes the volume serial number in the `VolumeSerialNumber` property in a hexadecimal string, while the `Win32_Volume`

ume class exposes the serial number in the *SerialNumber* property as a 32-bit integer. This is why Sample 3.47 includes some extra logic to display the volume serial number in a hexadecimal format (lines 466 through 469).

Sample 3.48

Executing the DefragAnalysis Win32_Volume method (Part VIII)

```
...:  
...:  
...:  
491:  
492:     ' -- Defragmentation Analysis -----  
493: If boolDefragAnalysis = True Then  
494:     WScript.Echo "Defragmentation analysis started ..."  
495:  
496:     intRC = objWMIInstance.DefragAnalysis (boolDefragRecommended, _  
497:                                               objDefragAnalysis)  
...:  
500:  
501:     If intRC Then  
502:         WScript.Echo "Defragmentation analysis error (" & inRC & ")."  
503:     Else  
504:         If boolDefragRecommended Then  
505:             WScript.Echo vbCRLF & "Defragmentation is recommended."  
506:         Else  
507:             WScript.Echo vbCRLF & "Defragmentation is NOT recommended."  
508:         End If  
509:         DisplayFormattedProperties objDefragAnalysis, 0  
510:     End If  
511: End If  
...:  
...:  
...:
```

The ability to request the defragmentation statistics and perform a defragmentation is a very interesting feature of the *Win32_Volume* class (see Sample 3.48 and 3.49). Both methods are very easy to use. Each of them relates to a specific instance of the *Win32_Volume* class. The *DefragAnalysis* method exposes two output parameters (lines 496 and 497): a Boolean variable informing if the defragmentation is recommended and an **SWbemObject** made from the *Win32_DefragAnalysis* class to return defragmentation statistics. The command line to use and the output obtained are as follows:

```
C:\>WMIDiskSvc.wsf D: /Action:DefragAnalysis  
Microsoft (R) Windows Script Host Version 5.6  
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
```

```
Defragmentation analysis started ...
```

```
Defragmentation is NOT recommended.
```

```
- Win32_DefragAnalysis -----  
AverageFileSize: ..... 141980  
AverageFragmentsPerFile: ..... 1
```

```

ClusterSize: ..... 4096
ExcessFolderFragments: ..... 0
FilePercentFragmentation: ..... 0
FragmentedFolders: ..... 1
FreeSpace: ..... 1724788736
FreeSpacePercent: ..... 23
FreeSpacePercentFragmentation: ..... 2
MFTPercentInUse: ..... 84
MFTRecordCount: ..... 55029
PageFileSize: ..... 0
TotalExcessFragments: ..... 360
TotalFiles: ..... 52188
TotalFolders: ..... 2790
TotalFragmentedFiles: ..... 12
TotalMFTFragments: ..... 2
TotalMFTSize: ..... 66368512
TotalPageFileFragments: ..... 0
TotalPercentFragmentation: ..... 1
UsedSpace: ..... 5497942016
VolumeSize: ..... 7222730752

```

→ **Sample 3.49** Executing the *Defrag Win32_Volume* method (Part IX)

```

...:
...:
...:
511:
512:      ' -- Defragmentation -----
513:      If boolDefrag = True Then
514:          WScript.Echo "Defragmentation started ..."
515:
516:          intRC = objWMIInstance.Defrag (boolForce, objDefragAnalysis)
...:
519:          If intRC Then
520:              WScript.Echo "Defragmentation error (" & inRC & ")."
521:          Else
522:              DisplayFormattedProperties objDefragAnalysis, 0
523:          End If
524:      End If
525:
...:
...:
...:
```

The invocation of the *Win32_Volume Defrag* method follows the same coding technique as the *DefragAnalysis* method (see Sample 3.49). The only difference resides in the method parameters, where one input parameter is required to eventually force the volume defragmentation and one output parameter is available to return the defragmentation analysis information (line 516). The execution of this code portion of the script is made with the following command line:

```
C:\>WMIDiskSvc.wsf D: /Action:Defrag
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
```

```
Defragmentation started ...

- Win32_DefragAnalysis -----
AverageFileSize: ..... 141980
AverageFragmentsPerFile: ..... 1
ClusterSize: ..... 4096
ExcessFolderFragments: ..... 0
FilePercentFragmentation: ..... 0
FragmentedFolders: ..... 1
FreeSpace: ..... 1724788736
FreeSpacePercent: ..... 23
FreeSpacePercentFragmentation: ..... 2
MFTPercentInUse: ..... 84
MFTRecordCount: ..... 55029
PageFileSize: ..... 0
TotalExcessFragments: ..... 16
TotalFiles: ..... 52188
TotalFolders: ..... 2790
TotalFragmentedFiles: ..... 1
TotalMFTFragments: ..... 2
TotalMFTSize: ..... 66368512
TotalPageFileFragments: ..... 0
TotalPercentFragmentation: ..... 1
UsedSpace: ..... 5497942016
VolumeSize: ..... 7222730752
```

With the *Format* method of the *Win32_Volume* class, it is possible to format a volume (see Sample 3.50). Basically, this method implements the functionality of the FORMAT.COM command. The *Format* method accepts five input parameters: the file system type (NTFS, FAT, or FAT32), a Boolean variable defining the formatting mode (true=quick or false=normal), the cluster size (by default 4,096), the volume label, and a Boolean variable to enable the compression (lines 530 through 534). The latter is actually not implemented, even if it is required to invoke the method.

→ **Sample 3.50** *Executing the Format Win32_Volume method (Part X)*

```
...:
...:
...:
525:
526:     ' -- Format -----
527:     If boolFormat = True Then
528:         WScript.Echo "Format started ..."
529:
530:         intRC = objWMIInstance.Format (strFileSystem, _
531:                                         boolQuickFormat, _
532:                                         intClusterSize, _
533:                                         strLabel, _
534:                                         boolCompression)
...:
537:     If intRC Then
538:         WScript.Echo "Format error (" & intRC & ")."
539:     Else
540:         WScript.Echo "Format completed successfully."
```

```

541:           End If
542:       End If
543:
...:
...:
...:
```

If a shadow storage volume is associated with a *Win32_Volume* instance, it is possible to update the maximum disk space that can be allocated to the shadow copies. To do so, it is necessary to retrieve the association instance made from the *Win32_ShadowStorage* class (see Sample 3.51). The execution of a WQL data query from the *Win32_Volume* instance representing the disk containing the information to shadow will retrieve the association *Win32_ShadowStorage* instance (lines 549 through 552). The WQL data query will be as follows for a given *Win32_Volume* instance:

```
References of {Win32_Volume.DeviceID="\\\\?\Volume{3eed7424-a3b2-11d6-a5f4-806e6f6e6963}\\"}
Where ResultClass=Win32_ShadowStorage
```

To create a *Win32_ShadowStorage* instance, Sample 3.45 (“Associating a shadow storage with a *Win32_Volume* instance [Part V]”) requests a drive letter and a disk space allocated to the shadows. Actually, the disk space value can be updated after the *Win32_ShadowStorage* instance creation by examining the *MaxSpace* property of the corresponding *Win32_ShadowStorage* instance (see Sample 3.51). This *Win32_ShadowStorage* instance can be retrieved with a WQL data query similar to the previous one (lines 549 through 552). As mentioned previously, there is only one instance of the *Win32_ShadowStorage* class per *Win32_Volume* instance. This is the reason why the script tests the number of results at line 555. Because a WQL data query returns the result in a collection, by enumerating the collection (lines 556 through 562), it is possible to update the *MaxSpace* property of the *Win32_ShadowStorage* instances (line 557). From the user interface (see Figure 3.24), it is possible to set no limit on the space allocated to the shadows. The script supports this as well by using the **Unlimited** keyword from the command line:

```

1:  C:\>WMIDiskSvc.wsf D: /Action:UpdateShadowStorage /MaxSpace:Unlimited
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Updating maximum space for the shadow volume ...
6:  Maximum space for Shadow Storage successfully updated.
```

Actually, the **Unlimited** keyword sets the *MaxSpace* property to a value equal to 18446744073709551615 (which corresponds to $2^{64} - 1$) defined as a constant at line 76 of Sample 3.51.

→ **Sample 3.51** *Updating a Win32_ShadowStorage instance (Part XI)*

```
...:
...:
...:
543:
544:     ' -- Update Shadow Storage -----
545: If boolUpdateShadowStorage = True Then
546:     If varMaxSpace <> -1 Then
547:         WScript.Echo "Updating maximum space for the shadow volume ..."
548:
549:         Set objWMIAssociatedInstances = objWMIServices.ExecQuery _
550:             ("References of {" &
551:                 objWMInstance.Path_.RelPath & _
552:                     "} Where ResultClass=Win32_ShadowStorage")
...:
555:     If objWMIAssociatedInstances.Count = 1 Then
556:         For Each objWMIAssociatedInstance In objWMIAssociatedInstances
557:             objWMIAssociatedInstance.MaxSpace = varMaxSpace
558:
559:             objWMIAssociatedInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
560:                                         wbemFlagReturnWhenComplete)
...:
562:             Next
563:
564:             WScript.Echo "Maximum space for Shadow Storage successfully updated."
565:         Else
566:             WScript.Echo "No Shadow Storage for volume '" & strWMIDriverLetter
567:         End If
568:     End If
569: End If
570:
...:
...:
...:
```

If we can create and update the *Win32_ShadowStorage* instances, it is possible to delete a *Win32_ShadowStorage* instance. The scripting technique is exactly the same as the update process, with the exception that the instance is deleted by invoking the *Delete_* method of the **SWBemObject** object representing the *Win32_ShadowStorage* instance (see Sample 3.52).

→ **Sample 3.52** *Removing a Win32_ShadowStorage instance (Part XII)*

```
...:
...:
...:
570:
571:     ' -- Delete -----
572: If boolDeleteShadowStorage = True Then
573:     WScript.Echo "Resetting Shadow Storage area on default volume '" & _
574:                     strWMIDriverLetter & "'."
575:
576:     Set objWMIAssociatedInstances = objWMIServices.ExecQuery _
577:         ("References of {" &
```

```
578:                     objWMIInstance.Path_.RelPath & _
579:                     " } Where ResultClass=Win32_ShadowStorage")
...
582:             If objWMIAssociatedInstances.Count = 1 Then
583:                 For Each objWMIAssociatedInstance In objWMIAssociatedInstances
584:                     objWMIAssociatedInstance.Delete_
...
586:             Next
587:
588:             WScript.Echo "Shadow Storage successfully reset."
589:         Else
590:             WScript.Echo "No Shadow Storage other than the default for volume " & _
591:                         strWMIDriverLetter & "."
592:         End If
593:     End If
594:     End If
595: End If
596: Next
597: End If
...
602: ]]>
603: </script>
604: </job>
605:</package>
```

With the Shadow Copy classes, it is possible to perform all tasks available from the user interface and update almost all properties exposed by the user interface (see Figure 3.24). The only setting that is not manageable from WMI is the schedule information.

3.6 Active Directory components providers

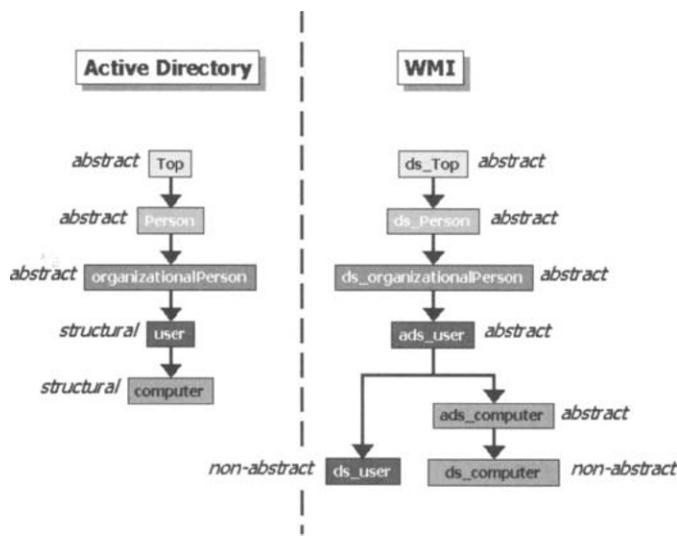
3.6.1 Active Directory Service providers

There are three *Active Directory* providers, as shown in Table 3.49.

Table 3.49 The Active Directory Providers Capabilities

To work with these providers, it is best to have an understanding of the Active Directory Schema, since WMI reflects its logical structure. All classes provided by the *Active Directory* class provider are mapped from the Active Directory schema classes. By accessing the `Root\Directory\LDAP` namespace, it is possible to reference any class and object in the Active Directory. Basically, we can say that the *Active Directory* providers mirror classes and instances from the Active Directory into this specific CIM repository namespace. To perform this mapping, the Active Directory Service providers follow naming rules to preserve the relationships that exist between the Active Directory classes and instances. Two mappings are realized: one for the Active Directory classes and one for the Active Directory instances.

Figure 3.26
The Active
Directory WMI
classes mapped to
the Active
Directory classes.

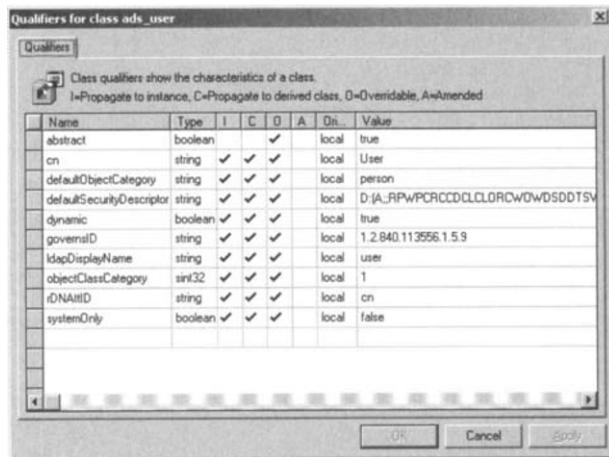


Let's take the Active Directory `user` class as an example. Defined in the Active Directory schema, the Active Directory `user` class is created from a class hierarchy starting from a Root class called `top` (see Figure 3.26, on the left side). To obtain the `user` class, several subclasses are created. This creation is called a derivation of classes, where the parent class is called a superclass. First, the `top` class is derived to obtain the `person` class. Next, the `person` class is derived to obtain the `organizationalPerson` class, which is in turn also derived to obtain the `user` class. Each class brings its set of Active Directory attributes. Each subclass inherits the set of attributes from the superclasses.

In Active Directory, the **user** class is defined as a structural class, which allows the creation of user instances from it. However, the **top**, **person**, and **organizationalPerson** classes are abstract classes in Active Directory and are used as parent templates to create their respective subclasses. As mentioned previously, classes in Active Directory are mapped to their equivalent WMI classes in the **Root\Directory\LDAP** namespace. In case of an abstract class, the WMI equivalent abstract class always uses the LDAP display name of the Active Directory class starting with the “*ds_*” prefix (Figure 3.26, on the right side). For example, for the Active Directory **organizationalPerson** class, we will have a corresponding *ds_organizationalPerson* WMI class. Since this Active Directory class is an abstract class, the WMI equivalent class is also an abstract class and has the *abstract* WMI qualifier set. When the Active Directory class is a non-abstract class (such as the **user** class, which is a structural class), the mapping is made to two WMI classes:

- A first class prefixed with the “*ads_*” prefix and implemented as a WMI abstract class (*abstract* qualifier is set). For the **user** class, we will have an *ads_user* WMI abstract class.

Figure 3.27
The WMI *ads_user* abstract class qualifiers.

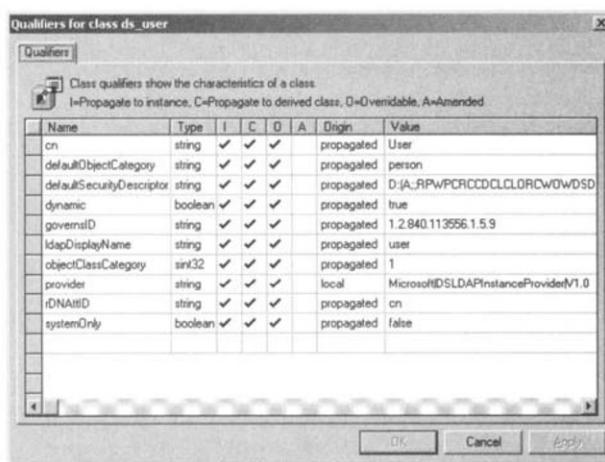


- A second class prefixed with the “*ds_*” prefix and implemented as a WMI dynamic instance class (provider qualifier is set). For the **user** class, we will have a *ds_user* WMI dynamic instance class.

In both Figures 3.27 and 3.28, you will notice the presence of other qualifiers representing Active Directory attributes defined in the schema and used to create the **user** class definition (i.e., **governsID**, **objectClassCategory**, **LDAPDisplayName**, etc.). In the same way, the syntax used by

Figure 3.28

The WMI *ds_user* dynamic instance class qualifiers.



Active Directory is also mapped to WMI. Table 3.50 shows the syntax mapping.

Table 3.50*The Active Directory/WMI Syntax Mapping*

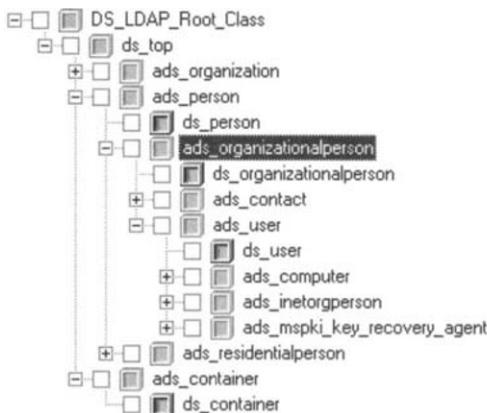
Active Directory syntax	WMI data type	WMI property value
Access-Point	CIM_STRING	Mapped from the value of the string.
Boolean	CIM_BOOLEAN	Mapped directly to the appropriate Boolean value.
Case Insensitive String	CIM_STRING	Mapped from the value of the string.
Case Sensitive String	CIM_STRING	Mapped from the value of the string.
Distinguished Name	CIM_STRING	Mapped from the value of the string.
DN-Binary	Embedded object of class DN_With_Binary	Mapped to instances of the DN_With_Binary class.
DN-String	Embedded object of class DN_With_String	Mapped to instances of the DN_With_String class.
Enumeration	CIM_SINT32	Mapped directly to the integer value.
IA5-String	CIM_STRING	Mapped from the value of the string.
Integer	CIM_SINT32	Mapped directly to the integer value.
NT Security Descriptor	Embedded object of Class Uint8Array	Mapped to instances of the Uint8Array class.
Numeric String	CIM_STRING	Mapped from the value of the string.
Object Id	CIM_STRING	Mapped from the string representation of the OID; for example, *1.3.3.4.*
Octet String	Embedded object of Class Uint8Array	Mapped to instances of the Uint8Array class.
OR Name	CIM_STRING	Mapped from the value of the string.
Presentation-Address	CIM_STRING	Mapped from the value of the string.
Print Case String	CIM_STRING	Mapped from the value of the string.
Replica Link	Embedded object of class Uint8Array	Mapped to instances of the Uint8Array class..
SID	Embedded object of Class Uint8Array	Mapped to instances of the Uint8Array class.
Time	CIM_DATETIME	Converted to the CIM_DATETIME representation and mapped.
Undefined	N/A	N/A
Unicode String	CIM_STRING	Mapped from the value of the string.
UTC Coded Time	CIM_DATETIME	Converted to the CIM_DATETIME representation and mapped.

The end result is that WMI exposes Active Directory classes as WMI classes (see Figure 3.29).

When a user object is created in Active Directory, it is always created in a container. The default container for user objects is the “Users” container, but it could be any other supported container, such as an organizational unit or a domain. In the Active Directory schema, the containers that can

Figure 3.29

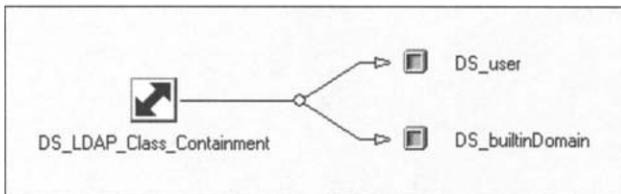
Some of the Active Directory classes as seen from WMI.



hold user objects are defined with the `possSuperiors` and `systemPossSuperiors` attributes of the Active Directory `user` class definition. These attributes reference the class (with their names) representing the supported containers—for example, the `domainDNS` class for the domain container or the `builtinDomain` class for the Users container. We see here that there is a relationship between the `user` class and the supported Active Directory container class definition. WMI represents this relationship with the `DS_LDAP_Class_Containment` association class (see Figure 3.30).

Figure 3.30

The DS_LDAP_Class_Containment associations.



In the same way, once user objects are instantiated, they are contained in an existing container. We will retrieve the same kind of relationship between the user objects and the containers but at the instance level instead of the class level. WMI represents this relation with the use of the `DS_LDAP_Instance_Containment` association class. (See Figure 3.31.)

These two association classes, along with the `RootDSE` WMI class, are listed in Table 3.51. The `RootDSE` WMI class represents the `RootDSE` LDAP object available from any LDAP v3 directory.

Every WMI instance representing an Active Directory object uses the Active Directory ADSI path of the object. The ADSI path is represented in the WMI class definition with the `ADSIPath` WMI Key property. For exam-

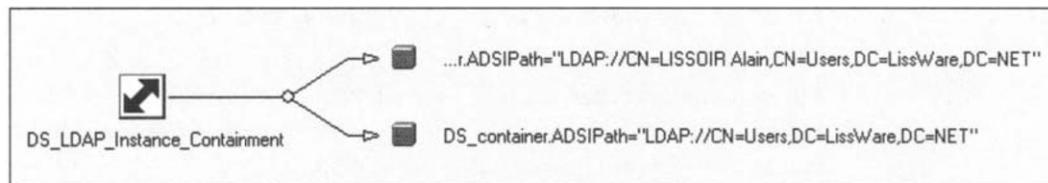


Figure 3.31 The *DS_LDAP_Instance_Containment* associations.

ple, the *ADSIPath* property for a user object called “LISSOIR Alain” and located in the “Users” containers of an Active Directory domain called Liss-Ware.Net will be:

```
LDAP://CN=LISSOIR Alain,CN=Users,DC=LissWare,DC=Net
```

There is one exception for the *RootDSE* WMI class, since it provides information about the capabilities of an LDAP server. This WMI class is a singleton class, since the *RootDSE* LDAP object is unique per LDAP server.

Everything we said about the *user* class and its WMI equivalents *ads_user* and *ds_user* WMI classes can be applied to any other class defined in the Active Directory schema. The logic is always the same. This is why a good understanding of the Active Directory schema mechanisms and definitions is helpful to navigate the WMI representation.

As seen in Table 3.49, the *Microsoft|DSLDAPInstanceProvider|V1.0* supports the “Get,” “Put,” “Enumeration,” and “Delete” operations. This clearly means that we can manipulate Active Directory object instances through WMI. With the previous script samples, we have seen how to create, update, and delete existing WMI instances. To manipulate the WMI Active Directory object instances there are no exceptions; the rules and the scripting techniques are exactly the same. However, because the *Microsoft|DSLDAPClassProvider|V1.0* only supports “Get” and “Enumeration” operations, it is not possible to create new classes in Active Directory with WMI. Therefore, it means we must create all Active Directory schema extensions with the Active Directory Service Interfaces (ADSI).

Table 3.51 The Active Directory Providers Classes

Name	Type	Comments
DS_LDAP_Class_Containment	Association	This class models the possible superiors of a DS class.
DS_LDAP_Instance_Containment	Association	This class models the parent-child container relationship of instances in the DS.
RootDSE	Dynamic	This is the class used to model the LDAP RootDSE object.

3.6.1.1 ***Creating and updating objects in Active Directory***

Although technically feasible with WMI to create, update, or delete Active Directory object instances, it is more efficient to use ADSI to perform these typical operations. As a basic example, we can create an Active Directory user with WMI, as illustrated in Sample 3.53.

Sample 3.53 *Creating an Active Directory user object with WMI*

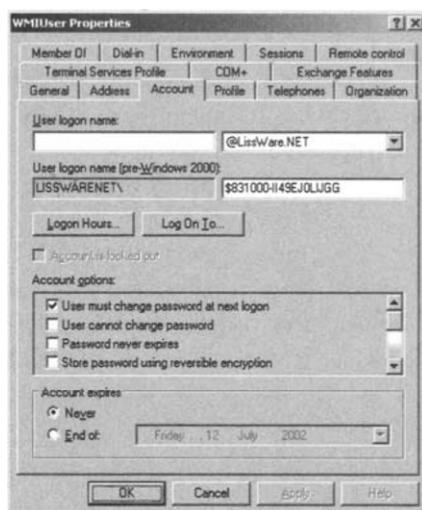
```
1:<?xml version="1.0"?>
.:
8:<package>
9: <job>
.:
13:   <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
14:
15:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:   <object progid="WbemScripting.SWbemNamedValueSet" id="objWMINamedValueSet" />
17:
18:   <script language="VBScript">
19:     <![CDATA[
.:
23:     Const cUserID = "WMIUser"
24:     Const cComputerName = "localhost"
25:     Const cWMINamespace = "Root/directory/LDAP"
26:     Const cWMIClass = "ds_user"
27:
28:     Const ADS_UF_ACCOUNTDISABLE = &h000002
.:
36:     objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
37:     objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
38:     Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINamespace, "", "")
.:
41:     Set objWMIClass = objWMIServices.Get (cWMIClass)
.:
44:     Set objWMIInstance = objWMIClass.SpawnInstance_
45:
46:     objWMIInstance.DS_sAMAccountName = cUserID
47:     objWMIInstance.ADSIPath = "LDAP://CN=" & cUserID & ",CN=Users,DC=LissWare,DC=Net"
48:
49:     objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or wbemFlagReturnWhenComplete)
.:
52:     WScript.Echo "Active Directory user successfully created."
53:
54:     objWMIInstance.Refresh_
55:
56:     objWMINamedValueSet.Add "__PUT_EXTENSIONS", True
57:     objWMINamedValueSet.Add "__PUT_EXT_CLIENT_REQUEST", True
58:     objWMINamedValueSet.Add "__PUT_EXT_PROPERTIES", Array ("DS_userAccountControl", _
59:                                         "DS_description")
60:
61:     objWMIInstance.DS_userAccountControl = objWMIInstance.DS_userAccountControl And _
62:                                         (NOT ADS_UF_ACCOUNTDISABLE)
63:     objWMIInstance.DS_description = Array ("Active Directory user created with WMI.")
64:
65:     objWMIInstance.Put_ wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete, _
66:                           objWMINamedValueSet
```

```
...  
69: WScript.Echo "Active Directory user successfully updated."  
...  
75: ]]>  
76: </script>  
77: </job>  
78:</package>
```

The technique to create a user instance in Active Directory with WMI is the same as any other instance creation. However, to completely demonstrate the instance creation and update techniques, Sample 3.53 is divided into two parts:

- The first part (lines 41 through 52) creates the WMI instance representing the Active Directory *user* object. As with any other instance creation, the code creates a new instance of the desired class (lines 41 and 44). Next, it assigns various properties required to create the new Active Directory *user* instance (lines 46 and 47). Although the *ADSI-Path* property is mandatory to create the new instance (since *Adsipath* is defined as a Key property of the *DS_user* WMI class in the CIM repository), the *DS_sAMAccountName* property is not required by WMI to create the new *user* instance. This may look amazing, since it is a mandatory attribute of the Active Directory *user* class, but, actually, if the *DS_sAMAccountName* value assignment is missing in the script, WMI generates a random value for it, as shown in Figure 3.32.

Figure 3.32
An Active
Directory user
created with WMI.



- The second part (lines 54 through 69) updates the previously created Active Directory **user** object. To update the existing Active Directory user, the script refreshes the instance information in order to get the latest information available from Active Directory (line 54), since some attributes are updated during the object creation (i.e., **modifyTimeStamp**, **userAccountControl**, **objectGUID**, **objectSID**, etc.). Under Windows 2000, since the *Refresh_* method is not available from an **SWBemObject** object, it is necessary to retrieve the created instance instead. Next, Sample 3.53 initializes an **SWBemNamedValueSet** object to perform a partial-instance update (lines 56 through 59). This partial-instance update updates two specific Active Directory attributes: the **userAccountControl** attribute (to enable the user object, since by default newly created users in Active Directory are disabled) and the **description** attribute (lines 61 through 63). The partial-instance update is mandatory, because saving the complete instance back to Active Directory will attempt to set some attributes that can only be set by the system, which means that the WMI call will fail to update the existing **user** object (lines 65 and 66). Once completed, the updated attributes are committed back by WMI to Active Directory.

The end result is the creation of an enabled Active Directory user with a specific description. Note that it is possible to commit the **description** attribute during the user creation (during the first part of the script). However, to enable the user, it is always necessary to create the user first and change its state by modifying the **userAccountControl** attribute next.

Although an Active Directory user creation is possible with WMI, it is clear that the ADSI scripting technique is more suitable for this type of task. However, there are situations where it could be useful to use the WMI technique to update a WMI instance representing an Active Directory object. As an example, this could be the case during WMI events notifying Active Directory objects creation. Once a user is created, WMI could detect its creation with a WQL event query and update the created Active Directory object with some information coming from a database (i.e., **telephoneNumber**, **postalAddress**, etc.). In such a case, the WQL query would be as follows:

```
1: C:\>GenericEventAsyncConsumer.wsf "Select * From __InstanceCreationEvent Within 5
   Where TargetInstance ISA 'DS_user'" /NameSpace:Root\Directory\LDAP
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
```

```

7: BEGIN - OnObjectReady.
8: Tuesday, 11 June, 2002 at 15:47:04: '__InstanceCreationEvent' has been triggered.
9: SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
10: TargetInstance (wbemCimtypeObject)
11: *ADSIPath (wbemCimtypeString) = LDAP://CN=Alain LISSOIR,CN=Users,DC=LissWare,DC=NET
12: DS_accountExpires (wbemCimtypeSint64) = 9223372036854775807
...
33: DS_badPasswordTime (wbemCimtypeSint64) = 0
34: DS_badPwdCount (wbemCimtypeSint32) = 0
...
41: DS_cn (wbemCimtypeString) = Alain LISSOIR
...
62: DS_displayName (wbemCimtypeString) = Alain LISSOIR
63: DS_displayNamePrintable (wbemCimtypeString) = (null)
64: DS_distinguishedName (wbemCimtypeString) = CN=Alain LISSOIR,CN=Users,DC=LissWare,DC=NET
65: DS_division (wbemCimtypeString) = (null)
66: DS_dLMemDefault (wbemCimtypeSint32) = (null)
67: DS_dLMemRejectPerms (wbemCimtypeString) = (null)
68: DS_dLMemRejectPermsBL (wbemCimtypeString) = (null)
69: DS_dLMemSubmitPerms (wbemCimtypeString) = (null)
70: DS_dLMemSubmitPermsBL (wbemCimtypeString) = (null)
...
...
...

```

3.6.1.2 Searching in Active Directory

To query Active Directory via LDAP, four key elements are required to formulate the query:

- The base object in which the search will start: This could be an object container in the Active Directory, such as an **organizationalUnit**

`OU=Brussels,DC=LissWare,DC=Net`

or a naming context, such as the **defaultNamingContext**, **Schema-NamingContext**.

`CN=Configuration,DC=LissWare,DC=Net`

`CN=Schema,CN=Configuration,DC=LissWare,DC=Net`

- The filter determines which elements have to be selected (based on conditions) from the Active Directory. Any accessible characteristics of an object can be used to make a query. Some examples are:

`(| (objectClass=domainDNS) (objectClass=organizationalUnit))`

This will return the list of all objects using the class **domainDNS** or **organizationalUnit**. Note the “**|**” sign for the “**or**” statement.

`(&(objectClass=user) (objectCategory=person))`

This will return the list of all objects using the class **user** for which the **objectCategory** is equal to **person**. Actually, this will provide a list of all users available in the selected naming context. In this example, the

objects of the class **contact** will be discarded, even if they are also using an **objectCategory** equal to **person**. Note the “**&**” sign for the “And” statement. To get both **contact** and **user** objects, the following syntax can be used:

```
(&(|(objectClass=user)(objectClass=contact))(objectCategory=person))
```

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

```
(objectCategory=person)
```

In the first version note the possibility to combine search conditions “**|**” and “**&**.”

- The attributes of the retrieved objects that are required. This can be any attribute associated with the retrieved objects (**cn**, **name**, **givenName**, **sn**, etc.).
- How deep the search has to run in the Active Directory.

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

OneLevel: A **OneLevel** search is restricted to the immediate children of a base object but excludes the base object itself. This scope is perfectly adapted for a search inside the **schemaNamingContext**.

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

To locate users residing in the Organizational Unit called “Brussels,” having a first name (**givenName**) equal to “Alain,” the following LDAP query filter must be used:

```
(&(objectclass=user)(givenName=Alain))
```

with the following base search:

```
OU=Brussels,DC=LissWare,DC=Net
```

and the following search level:

```
OneLevel
```

This LDAP query operation can be executed with the **LDP.Exe** utility available from the Windows 2000 or Windows Server 2003 Support Tools. The query result can be seen in Figure 3.33.

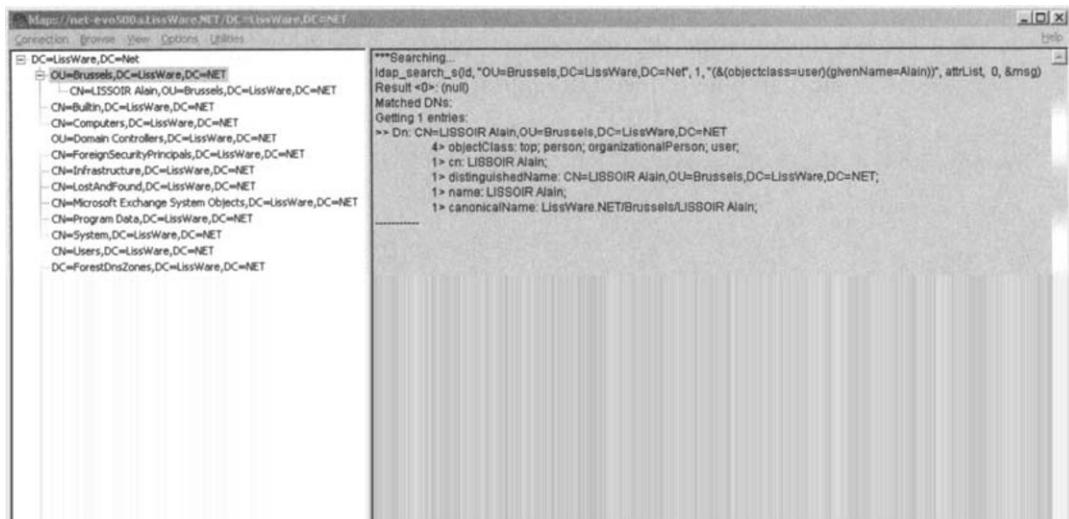


Figure 3.33 Querying Active Directory with LDAP from LDP.

Because WMI is connected to Active Directory, it is possible to obtain a similar result with a WQL data query:

```
Select * From DS_user Where DS_givenName='Alain'
```

However, there is no way to specify a base search. In this case, the search will be executed from the top of the domain tree (`DC=LissWare,DC=Net`). If a search must be performed in a context other than the Active Directory domain context (i.e., Configuration context), a *DN_Class* instance must be created. We will examine this technique later in this book, when monitoring the FSMO roles (see Sample 3.55, “Making the Configuration and Schema context accessible”).

3.6.1.3 Monitoring Active Directory group memberships

Everybody knows the importance of Active Directory group memberships, right? It is clear that some Active Directory groups are more sensitive than others from a security point of view. For example, you wouldn't want to see just anyone added to the “Enterprise Admins” group without appropriate control, would you? No, of course you wouldn't! Well, with the *Active Directory* provider and a WQL event query, it is possible to get a WMI event notification when a modification is made to a group or any other object of Active Directory. The WQL event query to use should be as follows:

```
Select * From __InstanceModificationEvent Within 10 Where TargetInstance ISA 'ds_group' AND  
TargetInstance.ds_name='Enterprise Admins'
```

Directly inspired from Samples 6.18 through 6.21 (“Monitoring, managing, and alerting script for the Windows services”) in the appendix , Sample 3.54 is an immediate application of this WQL Event query example.

Sample 3.54 *Monitoring, managing, and alerting script for the Windows Group modifications*

```

1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
18:   </runtime>
19:
20:   <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
21:   <script language="VBScript" src="..\Functions\PauseScript.vbs" />
22:   <script language="VBScript" src="..\Functions\GenerateHTML.vbs" />
23:   <script language="VBScript" src="..\Functions\SendMessageExtendedFunction.vbs" />
24:
25:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
26:   <object progid="WbemScripting.SWbemNamedValueSet" id="objWMIDateTimeSinkContext"/>
27:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
28:
29:   <script language="VBScript">
30:     <![CDATA[
.:
34:   '
35: Const cComputerName = "LocalHost"
36: Const cWMINameSpace = "Root/Directory/LDAP"
37: Const cWMIQuery = "Select * From __InstanceModificationEvent Within 10
                                         Where TargetInstance ISA 'ds_group' "
38:
39: Const cTargetRecipient = "Alain.Lissoir@LissWare.Net"
40: Const cSourceRecipient = "WMISystem@LissWare.Net"
41:
42: Const cSMTPServer = "10.10.10.3"
43: Const cSMTPPort = 25
44: Const cSMTPAccountName = ""
45: Const cSMTPSendEmailAddress = ""
46: Const cSMTPAuthenticate = 0' 0=Anonymous, 1=Basic, 2=NTLM
47: Const cSMTPUserName = ""
48: Const cSMTPPassword = ""
49: Const cSMTPSSL = False
50: Const cSMTPSendUsing = 2           ' 1=Pickup, 2=Port, 3=Exchange WebDAV
.:
67:   '
68:   ' Parse the command line parameters
69: If WScript.Arguments.Unnamed.Count = 0 Then
70:   WScript.Arguments.ShowUsage()
71:   WScript.Quit
72: Else
73:   For intIndice = 0 To WScript.Arguments.Unnamed.Count - 1
74:     ReDim Preserve strGroupName(intIndice)
75:     strGroupName(intIndice) = Ucase (WScript.Arguments.Unnamed.Item(intIndice))
76:   Next
77: End If

```

```
78:
79:     strUserID = WScript.Arguments.Named("User")
80:     If Len(strUserID) = 0 Then strUserID = ""
81:
82:     strPassword = WScript.Arguments.Named("strPassword")
83:     If Len(strPassword) = 0 Then strPassword = ""
84:
85:     strComputerName = WScript.Arguments.Named("Machine")
86:     If Len(strComputerName) = 0 Then strComputerName = cComputerName
87:
88:     Set objWMISink = WScript.CreateObject ("WbemScripting.SWbemSink", "SINK_")
89:
90:     objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
91:     objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
92:     Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINamespace, _
93:                                                 strUserID, strPassword)
94:
95:     For intIndice = 0 To UBound (strGroupName)
96:         If Len(strWMIQuery) = 0 Then
97:             strWMIQuery = "TargetInstance.ds_name=''" & strGroupName(intIndice) & "'"
98:         Else
99:             strWMIQuery = strWMIQuery & " Or " & _
100:                "TargetInstance.ds_name=''" & _
101:                strGroupName(intIndice) & "'"
102:
103:     End If
104:
105:     WScript.Echo "Adding '" & strGroupName(intIndice) & _
106:                 "' to subscription to monitor this Active Directory group."
107: Next
108:
109: strWMIQuery = cWMIQuery & " And (" & strWMIQuery & ")"
110:
111: objWMIServices.ExecNotificationQueryAsync objWMISink, strWMIQuery
112:
113: WScript.Echo vbCRLF & "Waiting for events..."
114:
115: PauseScript "Click on 'Ok' to terminate the script ..."
116:
117: WScript.Echo vbCRLF & "Cancelling event subscription ..."
118: objWMISink.Cancel
119:
120:
121: WScript.Echo "Finished."
122:
123: ' -----
124:
125:
126: Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)
127:
128:
129:     Wscript.Echo
130:     Wscript.Echo "BEGIN - OnObjectReady."
131:     Wscript.Echo FormatDateTime(Date, vbLongDate) & " at " & _
132:                               FormatDateTime(Time, vbLongTime) & ":" & _
133:                               objWbemObject.Path_.Class & "' has been triggered."
134:
135:     Select Case objWbemObject.Path_.Class
136:         Case "__InstanceModificationEvent"
137:             Set objWMIInstance = objWbemObject
138:
139:         Case "__AggregateEvent"
140:             Set objWMIInstance = objWbemObject.Representative
141:
142:         Case Else
143:             Set objWMIInstance = Null
144:
145:     End Select
```

```

147:
148:     If Not IsNull (objWMIInstance) Then
149:         If SendMessage (cTargetRecipient, _
150:                         cSourceRecipient, _
151:                         "" & objWMIInstance.TargetInstance.ds_name & _
152:                         " group modification - " & _
153:                         FormatDateTime(Date, vbLongDate) & _
154:                         " at " & _
155:                         FormatDateTime(Time, vbLongTime), _
156:                         GenerateHTML (objWMIInstance.PreviousInstance, _
157:                                         objWMIInstance.TargetInstance) , _
158:                                         "") Then
159:             WScript.Echo "Failed to send email to '" & cTargetRecipient & "' ..."
160:         End If
161:     End If
...
165:     Wscript.Echo "END - OnObjectReady."
166:
167: End Sub
168:
169: ]]>
170: </script>
171: </job>
172:</package>

```

As usual, the script structure is always the same. The script starts with the command-line parameter definitions (skipped lines 13 through 18) and parsing (lines 69 through 86). The script requires only one parameter on the command line, which lists one or more Active Directory group names to monitor. For example, to monitor modifications on the “Enterprise Admins” and “Domain Admins” group, the command line will be:

```
C:\>GroupMonitor.Wsf "Enterprise Admins" "Domain Admins"
```

The groups given on the command line are stored in an array (lines 75 through 78). Once the WMI connection is established (lines 90 through 93), the WQL query is constructed in a loop between lines 96 and 107. For the two sample groups given on the command line, the resulting WQL query will be:

```
Select * From __InstanceModificationEvent Within 10 Where TargetInstance ISA 'ds_group' And
(TargetInstance.ds_name='Enterprise Admins' Or TargetInstance.ds_name='Domain Admins')
```

Next, the WQL query is submitted for asynchronous notifications (line 111), and the script enters an idle state. Once a modification is made to one of the given groups, the event sink routine (lines 127 through 167) is invoked, and the script immediately sends an email alert by reusing the `SendMessage()` function (included at line 23) and the `GenerateHTML()` function (included at line 22). The `PreviousInstance` and the `TargetInstance` properties containing object instances are formatted in HTML and stored in a MIME body mail message.

3.6.1.4 Monitoring the FSMO roles

By using the event instrumentation provided by WMI, it is also possible to monitor the Domain Controllers Flexible Single Master Operations (FSMO) role modifications. However, this monitoring requires a small setup at the level of the `Root\directory\LDAP` namespace. By default, the *Active Directory* provider accesses any Active Directory object instances located in the default naming context, which is the Windows Domain where the accessed Domain Controller resides. However, to monitor all FSMO roles, it is necessary to access the Active Directory Configuration and Schema naming contexts, since the attributes containing the relevant information are spread among the different Active Directory naming contexts. Table 3.52 lists the different FSMO roles and their respective naming context locations.

Table 3.52 The Active FSMO Roles and Their Location in Active Directory

FSMO Role	Naming Context	WMI Class	Object distinguishedName	WMI Property
PDC Emulator	Domain	<code>ds_domaindns</code>	<code>DC=LissWare,DC=Net</code>	<code>ds_FSMORoleOwner</code>
Infrastructure Master	Domain	<code>ds_infrastructureupdate</code>	<code>CN=Infrastructure,DC=LissWare,DC=Net</code>	<code>ds_FSMORoleOwner</code>
RID Master	Domain	<code>ds_ridmanager</code>	<code>CN=RID Manager\$,CN=System,DC=LissWare,DC=Net</code>	<code>ds_FSMORoleOwner</code>
Domain Naming	Configuration	<code>ds_crossrefcontainer</code>	<code>CN=Partitions,CN=Configuration,DC=LissWare,DC=Net</code>	<code>ds_FSMORoleOwner</code>
Schema owner	Schema	<code>ds_dmd</code>	<code>CN=Schema,CN=Configuration,DC=LissWare,DC=Net</code>	<code>ds_FSMORoleOwner</code>

To enable access from WMI to the `ds_crossRefContainer` Active Directory object instance located in the Configuration naming context and to the `ds_dMD` Active Directory object instance located in the Schema naming context, an instance of the `DN_class` and the `DSClass_To_DNInstance` WMI classes first must be created in the `Root\directory\LDAP` namespace. This can be done with the help of a MOF file, as illustrated in Sample 3.55.

Sample 3.55 Making the Configuration and Schema context accessible

```

1: #pragma namespace ("\\\\.\\Root\\directory\\ldap")
2:
3: Instance of DN_Class
4:
5: {
6:     DN = "LDAP://CN=Configuration,DC= LissWare,DC=Net";
7: }
8:
9: Instance of DSClass_To_DNInstance
10:
11: {
12:     DSClass = "ds_crossRefContainer";
13:
14:     RootDNForSearchAndQuery = "DN_Class.DN=\"LDAP://CN=Configuration,DC= LissWare,DC=Net\"";
15: }
16:

```

```

17:
18:     Instance of DN_Class
19:
20:     {
21:         DN = "LDAP://CN=Schema,CN=Configuration,DC= LissWare,DC=Net";
22:     };
23:
24:     Instance of DSClass_To_DNInstance
25:
26:     {
27:         DSClass = "ds_dMD";
28:
29:         RootDNForSearchAndQuery =
30:             "DN_Class.DN=\\"LDAP://CN=Schema,CN=Configuration,DC=LissWare,DC=Net\\"";
31:     };

```

The *DSClass_To_DNInstance* is an association class performing the association of an instance defining an Active Directory naming context and a WMI class representing the Active Directory class of the object instance to locate in that naming context. For example, to locate a *ds_crossRefContainer* instance in the Active Directory Configuration naming context, an instance of the *DSClass_To_DNInstance* association class must be created (lines 9 through 15) with its properties initialized as follows:

- The *DSClass* property is assigned with the *ds_crossRefContainer* class instance name (line 12).
- The *RootDNForSearchAndQuery* property is assigned with the WMI path of the WMI instance representing the Active Directory Configuration context (line 14). The instance representing the Active Directory Configuration context is created with the *DN_Class* WMI class (lines 3 through 7).

The same rule applies for the *ds_dMD* class, which has its instance located in the Active Directory Schema naming context (lines 18 through 31). Once the MOF file is loaded in the CIM repository with MOF-COMP.EXE, the monitoring of the FSMO roles can take place. The logic is implemented in Sample 3.56.

Sample 3.56 *Monitoring, managing, and alerting script for the FSMO role modifications*

```

1:<?xml version="1.0"?>
.:
8:<package>
9:   <job>
.:
13:   <runtime>
.:
17:   </runtime>
18:
19:   <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
20:   <script language="VBScript" src="..\Functions\PauseScript.vbs" />

```

```
21: <script language="VBScript" src=..\Functions\GenerateHTML.vbs" />
22: <script language="VBScript" src=..\Functions\SendMessageExtendedFunction.vbs" />
23:
24: <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
25: <object progid="WbemScripting.SWBemNamedValueSet" id="objPDCFSMOSinkContext"/>
26: <object progid="WbemScripting.SWBemNamedValueSet" id="objINFFSMOSinkContext"/>
27: <object progid="WbemScripting.SWBemNamedValueSet" id="objRIDFSMOSinkContext"/>
28: <object progid="WbemScripting.SWBemNamedValueSet" id="objDOMFSMOSinkContext"/>
29: <object progid="WbemScripting.SWBemNamedValueSet" id="objSCHFSMOSinkContext"/>
30: <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
31:
32: <script language="VBScript">
33: <![CDATA[
...:
32: ' -----
33: ' Parse the command line parameters
34: strUserID = WScript.Arguments.Named("User")
35: If Len(strUserID) = 0 Then strUserID = ""
36:
37: strPassword = WScript.Arguments.Named("strPassword")
38: If Len(strPassword) = 0 Then strPassword = ""
39:
40: strComputerName = WScript.Arguments.Named("Machine")
41: If Len(strComputerName) = 0 Then strComputerName = cComputerName
42:
43: Set objWMISink = WScript.CreateObject ("WbemScripting.SWbemSink", "SINK_")
44:
45: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
46: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
47: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINamespace, _
48:                                                 strUserID, strPassword)
...
49: objPDCFSMOSinkContext.Add "FSMO", "PDC"
50: objWMIServices.ExecNotificationQueryAsync objWMISink, _
51:                                         CPDCFSMOWMIQuery, _
52:                                         ' -
53:                                         ' -
54:                                         ' -
55:                                         objPDCFSMOSinkContext
56:
57: If Err.Number Then ErrorHandler (Err)
58: WScript.Echo "Monitoring PDC FSMO role ..."
59:
60: objINFFSMOSinkContext.Add "FSMO", "INFRASTRUCTURE"
61: objWMIServices.ExecNotificationQueryAsync objWMISink, _
62:                                         CINFFSMOWMIQuery, _
63:                                         ' -
64:                                         ' -
65:                                         ' -
66:                                         objINFFSMOSinkContext
67:
68: If Err.Number Then ErrorHandler (Err)
69: WScript.Echo "Monitoring INFRASTRUCTURE FSMO role ..."
70:
71: objRIDFSMOSinkContext.Add "FSMO", "RID"
72: objWMIServices.ExecNotificationQueryAsync objWMISink, _
73:                                         CRIDFSMOWMIQuery, _
74:                                         ' -
75:                                         ' -
76:                                         ' -
77:                                         objRIDFSMOSinkContext
78:
79: If Err.Number Then ErrorHandler (Err)
```

```

119: WScript.Echo "Monitoring RID FSMO role ..."
120:
121: objDOMFSMOSinkContext.Add "FSMO", "DOMAIN NAMING"
122: objWMIServices.ExecNotificationQueryAsync objWMISink,
123:                                         cDOMFSMOWMIQuery, _
124:                                         ' -
125:                                         ' -
126:                                         ' -
127:                                         objDOMFSMOSinkContext
128: If Err.Number Then ErrorHandler (Err)
129: WScript.Echo "Monitoring DOMAIN NAMING FSMO role ..."
130:
131: objSCHFSMOSinkContext.Add "FSMO", "SCHEMA"
132: objWMIServices.ExecNotificationQueryAsync objWMISink,
133:                                         cSCHFSMOWMIQuery, _
134:                                         ' -
135:                                         ' -
136:                                         ' -
137:                                         objSCHFSMOSinkContext
138: If Err.Number Then ErrorHandler (Err)
139: WScript.Echo "Monitoring SCHEMA FSMO role ..."
140:
141: WScript.Echo vbCRLF & "Waiting for events..."
142:
143: PauseScript "Click on 'Ok' to terminate the script ..."
144:
145: WScript.Echo vbCRLF & "Cancelling event subscription ..."
146: objWMISink.Cancel
...
151: WScript.Echo "Finished."
152:
153: ' -----
154: Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)
...
203: End Sub
204:
205: ]]>
206: </script>
207: </job>
208:</package>

```

The logic and structure of Sample 3.56 is basically the same as Sample 3.54 (“Monitoring, managing, and alerting script for the Windows Group modifications”). However, instead of executing one WQL event query, Sample 3.56 executes five WQL event queries (one per FSMO role):

For the PDC Emulator FSMO:

```

Select * From __InstanceModificationEvent Within 10
  Where TargetInstance ISA 'ds_domaindns' And
    PreviousInstance.DS_fSMORoleOwner <> TargetInstance.DS_fSMORoleOwner

```

For the Infrastructure Master FSMO:

```

Select * From __InstanceModificationEvent Within 10
  Where TargetInstance ISA 'ds_infrastructureupdate' And
    PreviousInstance.DS_fSMORoleOwner <> TargetInstance.DS_fSMORoleOwner

```

For the RID Master FSMO:

```
Select * From __InstanceModificationEvent Within 10
  Where TargetInstance ISA 'ds_ridmanager' And
    PreviousInstance.DS_fSMORoleOwner <> TargetInstance.DS_fSMORoleOwner"
```

For the Domain Naming Master FSMO:

```
Select * From __InstanceModificationEvent Within 10
  Where TargetInstance ISA 'ds_crossrefcontainer' And
    PreviousInstance.DS_fSMORoleOwner <> TargetInstance.DS_fSMORoleOwner"
```

For the Schema Owner FSMO:

```
Select * From __InstanceModificationEvent Within 10
  Where TargetInstance ISA 'ds_dmd' And
    PreviousInstance.DS_fSMORoleOwner <> TargetInstance.DS_fSMORoleOwner"
```

Once the WMI connection is established (lines 85 through 88), the five WQL event queries are submitted to WMI (lines 91 through 139). Note that each WQL query makes use of a WMI context, since the same event sink subroutine is used to capture all WMI events. This routine has the exact same logic and structure as Sample 3.54 (“Monitoring, managing, and alerting script for the Windows Group modifications”).

3.6.1.5 Debugging Active Directory providers

If you experience trouble managing Active Directory objects with the WMI *Active Directory* providers, it is possible to trace the provider’s activity in a log file. The configuration of a registry key set activates the trace logging. The registry keys are located at:

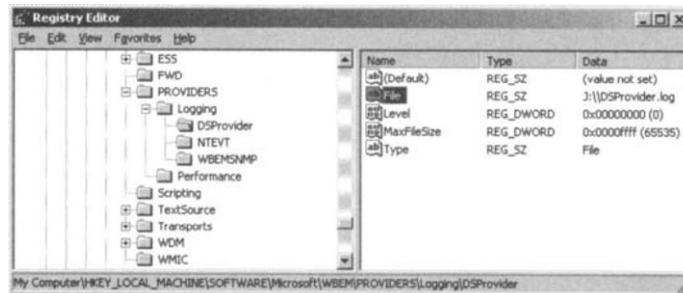
HKLM\SOFTWARE\Microsoft\WBEM\PROVIDERS\Logging\DSProvider

Table 3.53 Enabling the Trace Logging of a WMI Provider

Key names	Description
File	Full path and file name of the log file. The default value is %windir%\system32\wbem\logs. The Type named value must be set to “File” for this named value to be used.
Level	A 32-bit logical mask that defines the type of debugging output generated by the provider. This value is provider-dependent. The default value is 0 (zero).
MaxFileSize	Maximum file size (in bytes) of the log file. This integer value must be in the range 1024 to 2 ³² -1. When the file size exceeds this value, the file is renamed to -filename and a new, empty log file is created. The disk space required for the log file is twice the value of MaxFileSize. The default value is 65,535.
Type	Can be set to “File” or “Debugger”. If set to “File”, the trace information is written to the log file specified in the File named value. The default value is “File.”

Note that other WMI providers, such as *SNMP* providers, also support activity trace logging (see Table 3.53). They use the same set of registry key names but from a different registry hive, as shown in Figure 3.34; the *SNMP* providers use the “WBEMSNMP” hive.

Figure 3.34
The registry hive
for the four WMI
providers
supporting activity
logging.



Once the *Active Directory* WMI providers logging is started, the **DSProvider.LOG** file contains trace information and error messages for the *Directory Service* providers. The *Level* registry key is set to zero by default and can remain zero for the *Active Directory* providers. However, the tracing of other providers (i.e., *SNMP* providers) may require some values. Actually, the required values are determined by the provider implementation. To give a simple trace example, if the script (Sample 3.53, “Creating an Active Directory user object with WMI”) is executed a second time, it will return an error, since the user already exists. The output will be as follows:

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

-----
Error: &h80041001

Generic failure
-----
```

From a WMI perspective, the returned error message “Generic Failure” is not enough to determine the source of the problem. Of course, it is always possible to instantiate an **SWBemLastError** object, but in this particular case it will not give much more information about the problem. However, by looking at the **DSProvider.LOG**, it shows the following information:

```
1: CDSClassProvider :: GetClassFromCacheOrADSI()
Could not find class in Authenticated list for ds_user. Going to ADSI
2: CDSClassProvider :: GetObjectAsync() called for ADS_user
3: CDSClassProvider :: GetClassFromCacheOrADSI()
Could not find class in Authenticated list for ADS_user. Going to ADSI
4: CDSClassProvider :: GetObjectAsync() called for ADS_organizationalPerson
5: CDSClassProvider :: GetClassFromCacheOrADSI()
Could not find class in Authenticated list for ADS_organizationalPerson. Going to ADSI
6: CDSClassProvider :: GetObjectAsync() called for ADS_person
7: CDSClassProvider :: GetClassFromCacheOrADSI()
Could not find class in Authenticated list for ADS_person. Going to ADSI
8: CDSClassProvider :: GetObjectAsync() called for DS_top
```

```
9: CDSClassProvider :: GetClassFromCacheOrADSI()
   Could not find class in Authenticated list for DS_top. Going to ADSI
10: CWbemCache :: AddClass() Added a class DS_top to cache
11: CDSClassProvider :: GetClassFromCacheOrADSI()
   GetClassFromADSI succeeded for DS_top Added it to cache
12: CDSClassProvider :: GetClassFromCacheOrADSI() Also added to Authenticated list : DS_top
13: CWbemCache :: AddClass() Added a class ADS_person to cache
14: CDSClassProvider :: GetClassFromCacheOrADSI()
   GetClassFromADSI succeeded for ADS_person Added it to cache
15: CDSClassProvider :: GetClassFromCacheOrADSI()
   Also added to Authenticated list : ADS_person
16: CWbemCache :: AddClass() Added a class ADS_organizationalPerson to cache
17: CDSClassProvider :: GetClassFromCacheOrADSI()
   GetClassFromADSI succeeded for ADS_organizationalPerson Added it to cache
18: CDSClassProvider :: GetClassFromCacheOrADSI()
   Also added to Authenticated list : ADS_organizationalPerson
19: CWbemCache :: AddClass() Added a class ADS_user to cache
20: CDSClassProvider :: GetClassFromCacheOrADSI()
   GetClassFromADSI succeeded for ADS_user Added it to cache
21: CDSClassProvider :: GetClassFromCacheOrADSI() Also added to Authenticated list : ADS_user
22: CWbemCache :: AddClass() Added a class ds_user to cache
23: CDSClassProvider :: GetClassFromCacheOrADSI()
   GetClassFromADSI succeeded for ds_user Added it to cache
24: CDSClassProvider :: GetClassFromCacheOrADSI() Also added to Authenticated list : ds_user
25: CDSClassProvider :: GetObjectAsync() called for ds_user
26: CDSClassProvider :: GetClassFromCacheOrADSI() Found class in Authenticated list for ds_user
27: CDSClassProvider :: GetClassFromCacheOrADSI() Found class in cache for ds_user
28: CDSClassProvider :: GetObjectAsync() called for ads_user
29: CDSClassProvider :: GetClassFromCacheOrADSI() Found class in Authenticated list for ads_user
30: CDSClassProvider :: GetClassFromCacheOrADSI() Found class in cache for ads_user
31: CDSInstanceProviderClassFactory::CreateInstance() called
32: CLDAPInstanceProvider :: CONSTRUCTOR
33: CLDAPInstanceProvider :: Got Top Level Container as : DC=LissWare,DC=Net
34: CLDAPInstanceProvider :: PutInstanceAsync() called
35: CLDAPInstanceProvider :: PutInstanceAsync()
   calledfor ds_user.ADSIPath="LDAP://CN=WMIUser,CN=Users,DC=LissWare,DC=Net"
36: CLDAPInstanceProvider :: The 362 attributes being put are:
37: accountExpires
38: accountNameHistory
39: acSPolicyName
...
395: whenCreated
396: wWWHomePage
397: x121Address
398: x500uniqueIdentifier
399: CLDAPInstanceProvider :: SetObjectAttributes FAILED with 80072035
400: CLDAPInstanceProvider :: PutInstanceAsync()
   ModifyExistingInstance FAILED for LDAP://CN=WMIUser,CN=Users,DC=LissWare,DC=Net
   with 80072035
```

From line 1 through 30, we see the activity generated by the *Active Directory* provider to create the *ds_user* instance. At line 31, we see the instance creation followed by the *Put_* method invocation (lines 34 and 35). From line 36 through 398, we see the list of attributes that will be set. Because the user instance already exists in Active Directory, trying to set all these attributes will generate an error, since some attributes can only be set

by the system itself (see section 3.6.1.1, “Creating and updating objects in Active Directory”). The end result is an error number 80072035 (line 399). By looking in the Active Directory platform SDK, we can determine that an error 8007* is a Win32 Error. In such a case the rightmost part of the error number must be converted to a decimal value, which gives 8,245 in decimal (from 2,035 in hexadecimal). If we run the command “NET HELPMSG 8245” from the command line, we will get the informational message: “The server is unwilling to process the request.” This makes sense, since the script tries to perform an illegal Active Directory operation, which is an update of all attributes available from the existing user object (even with the ones that can only be updated by the system).

3.6.2 Active Directory Replication provider

The *Active Directory Replication* provider allows the management of the replication. With this provider and its supported classes, it is possible to retrieve information about the replication state. The provider is implemented as an instance and method provider (see Table 3.54).

Table 3.54 The Active Directory Replication Providers 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
Active Directory Replication provider																
ReplProv1	Root\Microsoft\ActiveDirectory	X	X				X	X	X							

The *Active Directory Replication* provider is located in the Root\Microsoft\ActiveDirectory namespace of the CIM repository and supports five classes, as listed in Table 3.55.

Table 3.55 The Active Directory Replication Providers Classes

Name	Comments
MSAD_ReplPendingOp	Describes a replication task currently executing or pending execution on the DC.
MSAD_NamingContext	Various properties of the current Naming Context.
MSAD_ReplCursor	Contains inbound replication state information with respect to all replicas of a given Naming Context. This state information indicates up to what USN X the destination server has seen all changes <= USN X originated by the source server with the given invocation ID.
MSAD_DomainController	The current domain controller properties.
MSAD_ReplNeighbor	Inbound replication state information for a Naming Context & source server pair.

Since the *Active Directory Replication* provider is not implemented as an event provider, there is no extrinsic event class available. This also means that the use of the **WITHIN** statement in a WQL event query is mandatory. For example, the *MSAD_DomainController* class exposes Boolean values that determine if a domain controller is registered in the Dynamic DNS or if the replicated SYSVOL volume is ready for use. To capture any modification made in an *MSAD_DomainController* instance that represents a domain controller, the following WQL event query can be used:

```
1: C:>GenericEventAsyncConsumer.wsf "Select * From __InstanceModificationEvent Within 10
2:                               Where TargetInstance ISA 'MSAD_DomainController'"
3:                               /namespace:Root\MicrosoftActiveDirectory
4: Microsoft (R) Windows Script Host Version 5.6
5: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
6:
7: Waiting for events...
8:
9: BEGIN - OnObjectReady.
10 Sunday, 16 December, 2001 at 10:07:11: '__InstanceModificationEvent' has been triggered.
11:
12: - __InstanceModificationEvent -----
13:
14: - MSAD_DomainController -----
15: CommonName: ..... NET-DPEN6400A
16: *DistinguishedName: ..... CN=NTDS Settings,CN=NET-DPEN6400A,CN=Servers,
17:                                         CN=Brussels,CN=Sites,CN=Configuration,
18:                                         DC=LissWare,DC=Net
19: IsAdvertisingToLocator: ..... TRUE
20: IsGC: ..... TRUE
21: IsNextRIDPoolAvailable: ..... FALSE
22: IsRegisteredInDNS: ..... FALSE
23: IsSysVolReady: ..... TRUE
24: NTDSaGUID: ..... 8b231f02-43e1-43f9-94c4-c8545e4b6d2b
25: PercentOfRIDsLeft: ..... 98
26: SiteName: ..... Brussels
27: TimeOfOldestReplAdd: ..... 01-01-1601
28: TimeOfOldestReplDel: ..... 01-01-1601
29: TimeOfOldestReplMod: ..... 01-01-1601
30: TimeOfOldestReplSync: ..... 01-01-1601
31: TimeOfOldestReplUpdRefs: ..... 01-01-1601
32:
33:
34: - MSAD_DomainController -----
35: CommonName: ..... NET-DPEN6400A
36: *DistinguishedName: ..... CN=NTDS Settings,CN=NET-DPEN6400A,CN=Servers,
37:                                         CN=Brussels,CN=Sites,CN=Configuration,
38:                                         DC=LissWare,DC=Net
39: IsAdvertisingToLocator: ..... TRUE
40: IsGC: ..... TRUE
41: IsNextRIDPoolAvailable: ..... FALSE
42: IsRegisteredInDNS: ..... TRUE
43: IsSysVolReady: ..... TRUE
44: NTDSaGUID: ..... 8b231f02-43e1-43f9-94c4-c8545e4b6d2b
45: PercentOfRIDsLeft: ..... 98
46: SiteName: ..... Brussels
47: TimeOfOldestReplAdd: ..... 01-01-1601
```

```

48:     TimeOfOldestReplDel: ..... 01-01-1601
49:     TimeOfOldestReplMod: ..... 01-01-1601
50:     TimeOfOldestReplSync: ..... 01-01-1601
51:     TimeOfOldestReplUpdRefs: ..... 01-01-1601
52:
53:
54: END - OnObjectReady.

```

In this output sample, we can see that the DNS registration state has passed from False (line 22) to True (line 42).

Samples 3.57 through 3.59 make use of the other classes supported by the *Active Directory Replication* provider (see Table 3.55). The script exposes the following command-line parameters:

```

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

Usage: WMIADRepl.wsf [/ReplPendingOp[+|-]] [/NC[+|-]] [/ReplCsr[+|-]] [/DC[+|-]]
        [/ReplNeighbor:value] [/ExecuteKCC[+|-]] [/SyncNC:value]
        [/Machine:value] [/User:value] [/Password:value]

```

Options:

ReplPendingOp	: View the replication tasks currently executing or pending execution on the DC.
NC	: View the properties of the current Naming Contexts.
ReplCsr	: View inbound replication state information with respect to all replicas of a each Naming Context.
DC	: View current domain controller properties.
ReplNeighbor	: View inbound replication state information for a Naming Context and source server pair.
ExecuteKCC	: Invokes the Knowledge Consistency Checker in order to verify the replication topology.
SyncNC	: Synchronizes a destination Naming Context with one of its sources.
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)

Example:

```

WMIADRepl.wsf /ReplPendingOp+
WMIADRepl.wsf /NC+
WMIADRepl.wsf /ReplCsr+
WMIADRepl.wsf /DC+
WMIADRepl.wsf /ReplNeighbor
WMIADRepl.wsf /ReplNeighbor:CN=Configuration,DC=LissWare,DC=Net
WMIADRepl.wsf /ReplNeighbor:CN=Schema,CN=Configuration,DC=LissWare,DC=Net
WMIADRepl.wsf /ReplNeighbor:DC=ForestDnsZones,DC=LissWare,DC=Net
WMIADRepl.wsf /ReplNeighbor:DC=Emea,DC=LissWare,DC=Net
WMIADRepl.wsf /ExecuteKCC+
WMIADRepl.wsf /SyncNC:CN=Configuration,DC=LissWare,DC=Net
WMIADRepl.wsf /SyncNC:CN=Schema,CN=Configuration,DC=LissWare,DC=Net
WMIADRepl.wsf /SyncNC:DC=ForestDnsZones,DC=LissWare,DC=Net
WMIADRepl.wsf /SyncNC:DC=Emea,DC=LissWare,DC=Net

```

Basically, the script enumerates the various instances of the classes.

→ **Sample 3.57 Viewing and managing the Active Directory Replication state (Part I)**

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
42:  </runtime>
43:
44:  <script language="VBScript" src=..\\Functions\\DisplayFormattedPropertyFunction.vbs" />
45:  <script language="VBScript" src=..\\Functions\\TinyErrorHandler.vbs" />
46:
47:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
48:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
49:
50:  <script language="VBScript">
51:  <![CDATA[
.:
55:  Const cComputerName = "LocalHost"
56:  Const cWMINameSpace = "Root/MicrosoftActiveDirectory"
57:
58:  Const cExecKCCTaskID = 0
59:  Const cExecKCCFlags = 0
60:  Const cSyncNCOOptions = 16
.:
87:  ' -----
88:  ' Parse the command line parameters
89:  If WScript.Arguments.Named.Count = 0 Then
90:      WScript.Arguments.ShowUsage()
91:      WScript.Quit
92:  End If
.:
121: strComputerName = WScript.Arguments.Named("Machine")
122: If Len(strComputerName) = 0 Then strComputerName = cComputerName
123:
124: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
125: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
126:
127: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
128:                                         strUserID, strPassword)
.:
131: ' -- ReplPendingOp -----
132: If boolReplPendingOp Then
133:     Set objWMIInstances = objWMIServices.InstancesOf ("MSAD_ReplPendingOp")
134:     If Err.Number Then ErrorHandler (Err)
135:
136:     If objWMIInstances.Count Then
137:
138:         WScript.Echo "~ " & objWMIInstance.NamingContextDN & " " & String (60, "-")
139:
140:         For Each objWMIInstance In objWMIInstances
141:             Set objWMIPropertySet = objWMIInstance.Properties_
142:             For Each objWMIProperty In objWMIPropertySet
143:                 DisplayFormattedProperty objWMIInstance, _
144:                               objWMIProperty.Name, _
```

```
145:                 objWMIProperty.Name, _  
146:                 Null  
147:             Next  
148:             Set objWMIPropertySet = Nothing  
149:             WScript.Echo  
150:         Next  
151:     Else  
152:         WScript.Echo "No information available." & vbCRLF  
153:     End If  
154: End If  
155:  
156: ' -- NC -----  
157: If boolNC Then  
158:     Set objWMIInstances = objWMIServices.InstancesOf ("MSAD_NamingContext")  
159:     If Err.Number Then ErrorHandler (Err)  
160:  
161:     WScript.Echo "EXISTING NAMING CONTEXT " & String (84, "=") & vbCRLF  
162:     WScript.Echo Space (36) & "Naming Context      Full replica"  
163:     WScript.Echo String (98, "-")  
164:  
165:     For Each objWMIInstance In objWMIInstances  
166:         WScript.Echo String (50 - Len (objWMIInstance.DistinguishedName), " ") & _  
167:                         objWMIInstance.DistinguishedName & " " & _  
168:                         String (15 - Len (objWMIInstance.IsFullReplica), " ") & _  
169:                         objWMIInstance.IsFullReplica  
170:     Next  
171:     WScript.Echo  
172: End If  
173:  
174: ' -- ReplCsr -----  
175: If boolReplCsr Then  
176:     Set objWMIInstances = objWMIServices.InstancesOf ("MSAD_ReplCursor")  
177:     If Err.Number Then ErrorHandler (Err)  
178:  
179:     If objWMIInstances.Count Then  
180:         For Each objWMIInstance In objWMIInstances  
181:             WScript.Echo "- " & objWMIInstance.NamingContextDN & " " & String (60, "-")  
182:  
183:             Set objWMIPropertySet = objWMIInstance.Properties_  
184:             For Each objWMIProperty In objWMIPropertySet  
185:                 DisplayFormattedProperty objWMIInstance, _  
186:                     objWMIProperty.Name, _  
187:                     objWMIProperty.Name, _  
188:                     Null  
189:             Next  
190:             Set objWMIPropertySet = Nothing  
191:             WScript.Echo  
192:         Next  
193:     Else  
194:         WScript.Echo "No information available." & vbCRLF  
195:     End If  
196: End If  
197:  
198: ' -- DC -----  
199: If boolDC Then  
200:     Set objWMIInstances = objWMIServices.InstancesOf ("MSAD_DomainController")  
201:     If Err.Number Then ErrorHandler (Err)  
202:  
203:     If objWMIInstances.Count Then  
204:         For Each objWMIInstance In objWMIInstances
```

```
205:         Set objWMIPropertySet = objWMIService.Properties_
206:         For Each objWMIProperty In objWMIPropertySet
207:             DisplayFormattedProperty objWMIService, _
208:                 objWMIProperty.Name, _
209:                 objWMIProperty.Name, _
210:                 Null
211:             Next
212:         Set objWMIPropertySet = Nothing
213:         WScript.Echo
214:     Next
215: Else
216:     WScript.Echo "No information available." & vbCRLF
217: End If
218: End If
219:
...:
...:
...:
```

Once the command-line parameters are parsed (lines 87 through 122) and the WMI connection established (lines 124 through 128), the script retrieves miscellaneous instances based on the command-line parameters. If the command line includes the **/ReplPendingOp+** switch, the section from line 132 through 154 is executed (see Sample 3.58). This section retrieves information about the replication tasks currently executing or pending execution on the Domain Controller by requesting all instances of the *MSAD_ReplPendingOp* class (line 133). Next, the script retrieves the properties of each instance (lines 140 through 150).

The script proceeds exactly in the same way for all classes supported by the *Active Directory Replication* provider. The script uses:

- The *MSAD_NamingContext* class to retrieve properties of the current Naming Context available (lines 157 through 172). A sample output would be:

```
1: C:\>WMIADRepl.wsf /NC+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: EXISTING NAMING CONTEXT =====
6:
7:                               Naming Context      Full replica
8: -----
9:           DC=DomainDnsZones,DC=LissWare,DC=Net          True
10:          DC=ForestDnsZones,DC=LissWare,DC=Net          True
11:          CN=Schema,CN=Configuration,DC=LissWare,DC=Net  True
12:          CN=Configuration,DC=LissWare,DC=Net            True
13:          DC=LissWare,DC=Net                            True
14:          DC=Emea,DC=LissWare,DC=Net                     False
15:
16: Completed.
```

- The *MSAD_ReplCursor* class to retrieve inbound replication state information with respect to all replicas of a given Naming Context (lines 175 through 196). A sample output would be:

```

1: C:\>WMIADRepl.wsf /ReplCsr+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - DC=DomainDnsZones,DC=LissWare,DC=Net -----
6: *NamingContextDN: ..... DC=DomainDnsZones,DC=LissWare,DC=Net
7: SourceDsaDN: ..... CN=NTDS Settings,CN=NET-DPEN6400A,CN=Servers,
8: ..... CN=Brussels,CN=Sites,CN=Configuration,
9: ..... DC=LissWare,DC=Net
10: *SourceDsaInvocationID: ..... 8b231f02-43e1-43f9-94c4-c8545e4b6d2b
11: TimeOfLastSuccessfulSync: ..... 16-12-2001 17:26:48
12: USNAttributeFilter: ..... 12212
13:
14: - DC=ForestDnsZones,DC=LissWare,DC=Net -----
15: *NamingContextDN: ..... DC=ForestDnsZones,DC=LissWare,DC=Net
16: SourceDsaDN: ..... CN=NTDS Settings,CN=NET-DPEN6400A,CN=Servers,
17: ..... CN=Brussels,CN=Sites,
18: ..... CN=Configuration,DC=LissWare,DC=Net
19: *SourceDsaInvocationID: ..... 8b231f02-43e1-43f9-94c4-c8545e4b6d2b
20: TimeOfLastSuccessfulSync: ..... 16-12-2001 17:26:48
21: USNAttributeFilter: ..... 12212

```

- The *MSAD_DomainController* class to retrieve the domain controller properties (lines 199 through 218). A sample output would be:

```

1: C:\>WMIADRepl.wsf /DC+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: CommonName: ..... NET-DPEN6400A
6: *DistinguishedName: ..... CN=NTDS Settings,CN=NET-DPEN6400A,CN=Servers,
7: ..... CN=Brussels,CN=Sites,CN=Configuration,
8: ..... DC=LissWare,DC=Net
9: IsAdvertisingToLocator: ..... TRUE
10: IsGC: ..... TRUE
11: IsNextRIDPoolAvailable: ..... FALSE
12: IsRegisteredInDNS: ..... TRUE
13: IsSysVolReady: ..... TRUE
14: NTDsaGUID: ..... 8b231f02-43e1-43f9-94c4-c8545e4b6d2b
15: PercentOfRIDsLeft: ..... 98
16: SiteName: ..... Brussels
17: TimeOfOldestReplAdd: ..... 01-01-1601
18: TimeOfOldestReplDel: ..... 01-01-1601
19: TimeOfOldestReplMod: ..... 01-01-1601
20: TimeOfOldestReplSync: ..... 01-01-1601
21: TimeOfOldestReplUpdRefs: ..... 01-01-1601
22:
23: Completed.

```

You will notice the presence of several Boolean values, which indicate the state of the examined domain controller (i.e., *IsAdvertisingToLocator*, *IsGC*, *IsNextRIDPoolAvailable*, *IsRegisteredInDNS*, *IsSysVolReady*).

Sample 3.58 displays information from the *MSAD_ReplNeighbor* class. It retrieves the inbound replication state information for a Naming Context and source server pair.

Sample 3.58

Viewing the inbound replication state information for a Naming Context (Part II)

```
...:  
...:  
...:  
219:  
220: ' -- ReplNeighbor -----  
221: If boolReplNeighbor Then  
222:   If Len (strReplNeighbor) Then  
223:     Set objWMIIInstances = objWMIServices.ExecQuery("Select * From MSAD_ReplNeighbor " &  
224:                                                 "Where NamingContextDN=" & _  
225:                                                 strReplNeighbor & "")  
226:   If Err.Number Then ErrorHandler (Err)  
227:  
228:   If objWMIIInstances.Count = 1 Then  
229:     For Each objWMIIInstance In objWMIIInstances  
230:  
231:       WScript.Echo "- " & objWMIIInstance.NamingContextDN & " " & String (60, "-")  
232:  
233:       Set objWMIPropertySet = objWMIIInstance.Properties_  
234:       For Each objWMIProperty In objWMIPropertySet  
235:         DisplayFormattedProperty objWMIIInstance, _  
236:           objWMIProperty.Name, _  
237:           objWMIProperty.Name, _  
238:           Null  
239:         Next  
240:       Set objWMIPropertySet = Nothing  
241:       WScript.Echo  
242:     Next  
243:   Else  
244:     WScript.Echo "No information available." & vbCRLF  
245:   End If  
246: Else  
247:   Set objWMIIInstances = objWMIServices.InstancesOf ("MSAD_ReplNeighbor")  
248:   If Err.Number Then ErrorHandler (Err)  
249:  
250:   If objWMIIInstances.Count Then  
251:     WScript.Echo "INBOUND REPLICATION STATE" & String (188, "=") & vbCRLF  
252:     WScript.Echo "                                         Naming Context" & _  
253:             "                 Source DSA          Site SyncProg" & _  
254:             " SyncNext IsDeleted LastSync ModSyncFailures SyncFailure" & _  
255:             "             TimeOfLastSync    TimeOfLastSuccess" & _  
256:             " USNAttr USNObject"  
257:     WScript.Echo String (213, "-")  
258:  
259:     For Each objWMIIInstance In objWMIIInstances  
260:       objWMIDateTime.Value = objWMIIInstance.TimeOfLastSyncAttempt  
261:       strTimeOfLastSyncAttempt = objWMIDateTime.GetVarDate (False)  
262:       objWMIDateTime.Value = objWMIIInstance.TimeOfLastSyncSuccess  
263:       strTimeOfLastSyncSuccess = objWMIDateTime.GetVarDate (False)  
264:  
265:     WScript.Echo String (50-Len(objWMIIInstance.NamingContextDN), " ") & _  
266:               objWMIIInstance.NamingContextDN & " " & _
```

```

267:             String (20-Len(objWMIInstance.SourceDsaCN), " ") & _
268:             objWMIInstance.SourceDsaCN & " " & _
269:             String (15-Len(objWMIInstance.SourceDsaSite), " ") & _
270:             objWMIInstance.SourceDsaSite & " " & _
271:             String (8-Len(objWMIInstance.FullSyncInProgress), " ") & _
272:             objWMIInstance.FullSyncInProgress & " " & _
273:             String (-Len(objWMIInstance.FullSyncNextPacket), " ") & _
274:             objWMIInstance.FullSyncNextPacket & " " & _
275:             String (9-Len(objWMIInstance.IsDeletedSourceDsa), " ") & _
276:             objWMIInstance.IsDeletedSourceDsa & " " & _
277:             String (8-Len(objWMIInstance.LastSyncResult), " ") & _
278:             objWMIInstance.LastSyncResult & " " & _
279:             String(15-Len(objWMIInstance.ModifiedNumConsecutiveSyncFailures), " ") & _
280:             objWMIInstance.ModifiedNumConsecutiveSyncFailures & " " & _
281:             String (11-Len(objWMIInstance.NumConsecutiveSyncFailures), " ") & _
282:             objWMIInstance.NumConsecutiveSyncFailures & " " & _
283:             String (20-Len(strTimeOfLastSyncAttempt), " ") & _
284:             strTimeOfLastSyncAttempt & " " & _
285:             String (20-Len(strTimeOfLastSyncSuccess), " ") & _
286:             strTimeOfLastSyncSuccess & " " & _
287:             String (8-Len(objWMIInstance.USNAttributeFilter), " ") & _
288:             objWMIInstance.USNAttributeFilter & " " & _
289:             String (9-Len(objWMIInstance.USNLastObjChangeSynced), " ") & _
290:             objWMIInstance.USNLastObjChangeSynced

291:         Next
292:         WScript.Echo
293:     Else
294:         WScript.Echo "No information available." & vbCrLf
295:     End If
296: End If
297: End If
298:
...:
...:
...:
```

The script can display the information in two different ways. If no specific naming context is specified with the **/ReplNeighbor** switch, the script displays the information for all naming contexts available (lines 247 through 296). If a naming context is given on the command line, only the information related to that naming context is displayed (lines 223 through 245). A sample output would be:

```

1: C:\>WMIADRepl.wsf /ReplNeighbor:CN=Configuration,DC=LissWare,DC=Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - CN=Configuration,DC=LissWare,DC=Net -----
6: AsyncIntersiteTransportObjGuid: ..... 00000000-0000-0000-0000-000000000000
7: CompressChanges: ..... TRUE
8: DisableScheduledSync: ..... FALSE
9: Domain: ..... LissWare.Net
10: DoScheduledSyncs: ..... TRUE
11: FullSyncInProgress: ..... FALSE
12: FullSyncNextPacket: ..... FALSE
13: IgnoreChangeNotifications: ..... FALSE
14: IsDeletedSourceDsa: ..... FALSE
15: LastSyncResult: ..... 0
```

```

16: ModifiedNumConsecutiveSyncFailures: ..... 0
17: *NamingContextDN: ..... CN=Configuration,DC=LissWare,DC=Net
18: NamingContextObjGuid: ..... ba172143-2bc0-4d55-be69-711a8f927419
19: NeverSynced: ..... FALSE
20: NoChangeNotifications: ..... TRUE
21: NumConsecutiveSyncFailures: ..... 0
22: ReplicaFlags: ..... 805306448
23: SourceDsaAddress: ..... 14612935-967d-402f-b5b1-0ae412edaec4.
24: ..... _msdcs.LissWare.Net
25: SourceDsaCN: ..... NET-DPEP6400
26: SourceDsaDN: ..... CN=NTDS Settings,CN=NET-DPEP6400,CN=Servers,
27: ..... CN=Seattle,CN=Sites,CN=Configuration,
28: ..... DC=LissWare,DC=Net
29: SourceDsaInvocationID: ..... 905e62c2-6f20-4333-ae8b-c6829e12d5b9
30: *SourceDsaObjGuid: ..... 14612935-967d-402f-b5b1-0ae412edaec4
31: SourceDsaSite: ..... Seattle
32: SyncOnStartup: ..... FALSE
33: TimeOfLastSyncAttempt: ..... 16-12-2001 17:38:05
34: TimeOfLastSyncSuccess: ..... 16-12-2001 17:38:05
35: TwoWaySync: ..... FALSE
36: UseAsyncIntersiteTransport: ..... FALSE
37: USNAttributeFilter: ..... 5810
38: USNLastObjChangeSynced: ..... 5810
39: Writeable: ..... TRUE
40:
41: Completed.

```

The class *MSAD_DomainController* exposes an interesting method called *ExecuteKCC*, which forces the execution of the Knowledge Consistency Checker (KCC). The *MSAD_DomainController* class exposing the *ExecuteKCC* method is a wrapper of the *DsReplicaConsistencyCheck()* API. Therefore, its method uses the same parameters as the API (see Table 3.56). These parameters are defined in constants at lines 58 and 59 (see Sample 3.57).

Table 3.56 The *ExecuteKCC* Method Parameters

Parameter name	Values	Description
TaskID		
DS_KCC_TASKID_UPDATE_TOPOLOGY	0	Identifies the task the KCC should execute. DS_KCC_TASKID_UPDATE_TOPOLOGY is the only supported value at this time.
Flags		
DS_KCC_FLAG_ASYNC_OP	1	DS_KCC_FLAG_ASYNC_OP. If specified, the server returns immediately, rather than waiting for the consistency check to complete.

To force the KCC execution, the script command line would be as follows:

```

1: C:\>WMIADRepl.wsf /ExecuteKCC+
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: KCC execution requested.
6:
7: Completed.

```

Another interesting method is the *SyncNamingContext* method exposed by the *MSAD_ReplNeighbor* class. With this method it is possible to force the replication of a specific Active Directory naming context. The *MSAD_ReplNeighbor* class exposing the *SyncNamingContext* method is a wrapper of the *DsReplicaSync()* API. Therefore, the method requires the same parameters as the API (see Table 3.57). These parameters are defined in constants at line 60 (see Sample 3.57).

Table 3.57*The SyncNamingContext Method Parameters*

Parameter name	Values	Description
Options		
DS_REPSYNC_ASYNCHRONOUS_OPERATION	0x1	Perform this operation asynchronously. Required when using DS_REPSYNC_ALL_SOURCES.
DS_REPSYNC_WRITEABLE	0x2	Writeable replica. Otherwise, read-only.
DS_REPSYNC_PERIODIC	0x4	This is a periodic sync request as scheduled by the admin.
DS_REPSYNC_INERSITE_MESSAGING	0x8	Use intersite messaging.
DS_REPSYNC_ALL_SOURCES	0x10	Sync from all sources.
DS_REPSYNC_FULL	0x20	Sync starting from scratch (i.e., at the first USN).
DS_REPSYNC_URGENT	0x40	This is a notification of an update that was marked urgent.
DS_REPSYNC_NO_DISCARD	0x80	Don't discard this synchronization request, even if a similar sync is pending.
DS_REPSYNC_FORCE	0x100	Sync even if link is currently disabled.
DS_REPSYNC_ADD_REFERENCE	0x200	Causes the source DSA to check if a rep-to is present for the local DSA source sends change notifications (aka the destination). If not, one is added. This ensures that source sends change notifications.

To force a naming context replication, the script command line would be as follows:

```
C:\>WMIADRepl.wsf /SyncNC:CN=Configuration,DC=LissWare,DC=Net
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Naming Context 'CN=Configuration,DC=LissWare,DC=Net' synchronization requested.

Completed.
```

The use of these two methods is shown in Sample 3.59. Lines 300 through 312 illustrate the *ExecuteKCC* method invocation, while lines 315 through 329 show the *SyncNamingContext* method invocation.

Sample 3.59*Triggering the KCC and forcing a Naming Context replication (Part III)*

```
...:
...:
...:
298:
299:   ' - ExecuteKCC -----
300:   If boolExecuteKCC Then
301:     Set objWMIIInstances = objWMIServices.InstancesOf ("MSAD_DomainController")
...:
304:   If objWMIIInstances.Count = 1 Then
305:     For Each objWMIIInstance In objWMIIInstances
306:       objWMIIInstance.ExecuteKCC cExecKCCTaskID, cExecKCCFlags
```

```
307:      Next
308:      WScript.Echo "KCC execution requested." & vbCrLf
309:      Else
310:          WScript.Echo "No information available." & vbCrLf
311:      End If
312:  End If
313:
314:  ' -- Sync -----
315:  If boolSyncNC Then
316:      Set objWMIInstances = objWMIServices.ExecQuery ("Select * From MSAD_ReplNeighbor " &
317:                                         "Where NamingContextDN=" & _
318:                                         strSyncNC & "'")
319:
320:
321:  If objWMIInstances.Count = 1 Then
322:      For Each objWMIInstance In objWMIInstances
323:          objWMIInstance.SyncNamingContext cSyncNCOOptions
324:      Next
325:      WScript.Echo "Naming Context '" & strSyncNC & "' synchronization requested." & vbCrLf
326:  Else
327:      WScript.Echo "No corresponding context to synchronize." & vbCrLf
328:  End If
329: End If
330:
331: WScript.Echo "Completed."
332:
333: ]]>
334: </script>
335: </job>
336:</package>
```

Note the scripting technique used to retrieve the naming context to synchronize (lines 316 through 318). The script executes a WQL data query with the naming context given on the command line to locate its corresponding WMI instance.

3.7 Network components providers

3.7.1 Ping provider

The *Ping* provider is a bit unusual, because it only supports the “Get” operation. Trying an “enumeration” operation makes no sense, since this provider reflects the result of the PING command execution. This provider only supports one class: the *Win32_PingStatus* class. (See Table 3.58.)

Creating an instance of the *Win32_PingStatus* class to ping a host is not really useful, but combining this functionality with some other WMI capabilities can produce a useful tool to determine if a system is still alive on the network. If we use a script code to create and delete an Interval Timer, combined with some asynchronous event notifications, we can develop a script that will ping a selected host on a regular time interval basis. Moreover, we can send an email alert to the Administrator if the PING reply is not suc-

Table 3.58 The Ping Providers 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
PING Provider																
WMI PingProvider	Root/CIMv2	X				X	X	X	X							

cessful by reusing some of the function previously developed (i.e., Send-Alert() function). The script reflects the PING.Exe command-line utility with some extra parameters specific to the script functionalities. The script command-line parameters are as follows:

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

Usage: WMIPingMonitor.wsf host [/a[+|-]] [/l:value] [/f[+|-]] [/i:value] [/v:value] [/r:value]
      [/s:value] [/j:value] [/k:value] [/w:value] [/interval:value]
      [/verbose[+|-]] [/Alert[+|-]] [/Machine:value] [/User:value]
      [/Password:value]
```

Options:

```
host    : IP Address or host name of the system to ping.
a      : Resolve addresses to hostnames.
l      : Send buffer size (size).
f      : Set Don't Fragment flag in packet.
i      : Time To Live (TTL).
v      : Type Of Service (TOS) (0=default, 2=Minimize Monetary Cost,
        4=Maximize Reliability, 8=Maximize Throughput, 16=Minimize Delay).
r      : Record route for count hops (count).
s      : Timestamp for count hops (count).
j      : Loose source route along host-list (host-list).
k      : Strict source route along host-list (host-list).
w      : Timeout in milliseconds to wait for each reply (timeout).
interval : Time interval between PING commands (in seconds).
verbose  : Show all Win32_PingStatus properties.
Alert    : Send an email in case of PING failure.
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)
```

Examples:

```
WMIPingMonitor.wsf 10.10.10.1
WMIPingMonitor.wsf 10.10.10.1 /a+ /l:32 /f+ /i:80 /v:4 /r:2 /s:1
WMIPingMonitor.wsf 10.10.10.1 /a+ /l:32 /f+ /i:80 /v:4 /r:5 /s:1 /Interval:10
WMIPingMonitor.wsf 10.10.10.1 /a+ /l:32 /f+ /i:80 /v:4 /r:6 /s:1 /Interval:10 /Verbose+
WMIPingMonitor.wsf 10.10.10.1 /a+ /l:32 /f+ /i:80 /v:4 /r:6 /s:1 /Verbose+ /Alert-
WMIPingMonitor.wsf 10.10.10.1 /j "10.10.10.254,192.1.1.1"
```

```
WMIPingMonitor.wsf 10.10.10.1 /k "10.10.10.254,192.1.1.1"  
WMIPingMonitor.wsf 10.10.10.1 /a+ /w:10000  
WMIPingMonitor.wsf 10.10.10.1 /a+ /Machine:NET-DOPEN6400A.LissWare.Net  
                                /Account:LISSWARENET\Administrator /Password:password
```

If we use the following command-line parameters:

```
C:\>WMIPingMonitor.wsf 16.174.12.1 /a+ /l:32 /f+ /i:80 /v:4 /r:3 /s:1 /verbose+
```

we will ping a host with the 16.174.12.1 IP address, request a host name resolution (*/a+*), use a packet size of 32 bytes (*/l:32*), ensure that the packet is not fragmented (*/f+*), request a time to live (TTL) of 80 seconds, request a service for a maximum reliability (*/v:4*), limit the record route for count hops to 3 (*/r:3*) and the timestamp for count hops to 1 (*/s:1*). The presence of the */Verbose+* switch forces the script to display all properties of the *Win32_PingStatus* class. The obtained output would be as follows:

```
1: C:\>WMIPingMonitor.wsf 16.174.12.1 /a+ /l:32 /f+ /i:80 /v:4 /r:3 /s:1 /verbose+  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5: PING timer created.  
6: Waiting for events...  
7:  
8: Sunday, 09 December, 2001 at 18:38:29.  
9: WMIPPING host '16.174.12.1' started ...  
10: Host '16.174.12.1' successfully contacted.  
11:  
12: - Win32_PingStatus -----  
13: *Address: ..... 16.174.12.1  
14: *BufferSize: ..... 32  
15: *NoFragmentation: ..... TRUE  
16: PrimaryAddressResolutionStatus: ..... 0  
17: ProtocolAddress: ..... 16.174.12.1  
18: ProtocolAddressResolved: ..... NET-DPEP6400A.Emea.LissWare.Net  
19: *RecordRoute: ..... 3  
20: ReplyInconsistency: ..... FALSE  
21: ReplySize: ..... 32  
22: *ResolveAddressNames: ..... TRUE  
23: ResponseTime: ..... 44  
24: ResponseTimeToLive: ..... 125  
25: RouteRecord: ..... 16.183.44.1  
26: ..... 16.183.16.1  
27: ..... 16.174.12.1  
28: RouteRecordResolved: ..... NET-DOPEN6400A  
29: ..... 16.183.16.1  
30: ..... NET-DPEP6400A.Emea.LissWare.Net  
31: *SourceRoute: .....  
32: *SourceRouteType: ..... 0  
33: StatusCode: ..... 0  
34: *Timeout: ..... 4000  
35: TimeStampRecord: ..... -1513502205  
36: TimeStampRecordAddress: ..... 16.183.44.1  
37: TimeStampRecordAddressResolved: ..... NET-DOPEN6400A  
38: *TimestampRoute: ..... 1
```

```
39: *TimeToLive: ..... 80
40: *TypeofService: ..... 80
41:
42: WMIPING host '16.174.12.1' ended.
```

The *Win32_PingStatus* class is only available from the Windows XP and Windows Server 2003 platforms. Therefore, it is possible to request a PING from one of these platforms to any other host on the network. Note that you can perform a WMI remote connection from any WMI-enabled system to a Windows XP or a Windows Server 2003 platform. From there the host targeted by the PING command can be any TCP/IP host. The key point is to instantiate the *Win32_PingStatus* class from a Windows XP or Windows Server 2003 platform. In this case, we obtain a PING command that is remotely executed—an interesting feature compared with the traditional PING command-line utility, which is, by default, executed locally.

The *Win32_PingStatus* class usage with an asynchronous event timer is shown in Sample 3.60. The command-line parameter definition and parsing sections are skipped, since they continue to use the same logic.

→ **Sample 3.60** *PINGing a system at regular time intervals (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
44:   </runtime>
45:
46:   <script language="VBScript" src=".\\Functions\\DecodeWinSockAPIErrorsFunction.vbs" />
47:   <script language="VBScript" src=".\\Functions\\DecodePINGStatusCodeFunction.vbs" />
48:
49:   <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertiesFunction.vbs" />
50:   <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
51:   <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
52:   <script language="VBScript" src=".\\Functions\\PauseScript.vbs" />
53:   <script language="VBScript" src=".\\Functions\\SendAlertFunction.vbs" />
54:   <script language="VBScript" src=".\\Functions\\SendMessageExtendedFunction.vbs" />
55:
56:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
57:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
58:   <object progid="Microsoft.XMLDOM" id="objXML" />
59:   <object progid="Microsoft.XMLDOM" id="objXSL" />
60:
61:   <script language="VBScript">
62:     <![CDATA[
.:
66: Const cComputerName = "LocalHost"
67: Const cWMINameSpace = "Root/cimv2"
68: Const cWMITimerClass = "__IntervalTimerInstruction"
69: Const cPINGTimerID = "MyPINGTimerEvent"
```

```
70: Const cWMIPINGEventQuery = "Select * From __TimerEvent Where TimerID='MyPINGTimerEvent'"  
71: Const cWMIPingStatusClass = "Win32_PingStatus"  
72:  
73: Const cTargetRecipient = "Alain.Lissoir@LissWare.Net"  
74: Const cSourceRecipient = "WMISystem@LissWare.Net"  
75:  
76: Const cXSLFile = "PathLevel0Win32_PingStatus.XSL"  
77:  
78: Const cSMTPServer = "relay.LissWare.Net"  
79: Const cSMTPPort = 25  
80: Const cSMTPAccountName = ""  
81: Const cSMTPSendEmailAddress = ""  
82: Const cSMTPAuthenticate = 0' 0=Anonymous, 1=Basic, 2=NTLM  
83: Const cSMTPUserName = ""  
84: Const cSMTPPassword = ""  
85: Const cSMTPSSL = False  
86: Const cSMTPSendUsing = 2           ' 1=Pickup, 2=Port, 3=Exchange WebDAV  
...:  
115: '  
116: ' Parse the command line parameters  
117: If WScript.Arguments.UnNamed.Count <> 1 Then  
118:     WScript.Arguments.ShowUsage()  
119:     WScript.Quit  
120: End If  
121:  
122: strHost = WScript.Arguments.Unnamed.Item(0)  
123: If Len(strHost) = 0 Then  
124:     WScript.Echo "Missing host address!"  
125:     WScript.Arguments.ShowUsage()  
126:     WScript.Quit  
127: End If  
128:  
129: boolResolveAddressNames = WScript.Arguments.Named("a")  
130: If Len (boolResolveAddressNames) = 0 Then boolResolveAddressNames = False  
131:  
...:  
177: strComputerName = WScript.Arguments.Named("Machine")  
178: If Len(strComputerName) = 0 Then strComputerName = cComputerName  
179:  
180: Set objWMISink = WScript.CreateObject ("WbemScripting.SWbemSink", "SINK_")  
181:  
182: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault  
183: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate  
184:  
185: Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINamespace, _  
186:                                         strUserID, strPassword)  
...:  
189: '  
190: Set objWMIClass = objWMIServices.Get (cWMITimerClass)  
191:  
192: Set objWMIInstance = objWMIClass.SpawnInstance_  
193: objWMIInstance.TimerID = cPINGTimerID  
194: objWMIInstance.IntervalBetweenEvents = intInterval * 1000  
195: objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or wbemFlagReturnWhenComplete)  
196: If Err.Number Then ErrorHandler (Err)  
197:  
198: WScript.Echo "PING timer created."  
199:  
200: '  
201: objWMIServices.ExecNotificationQueryAsync objWMISink, cWMIPINGEventQuery
```

```

202: If Err.Number Then ErrorHandler (Err)
203:
204: WScript.Echo "Waiting for events..."
205:
206: PauseScript "Click on 'Ok' to terminate the script ..."
207:
208: WScript.Echo vbCRLF & "Cancelling event subscription ..."
209: objWMISink.Cancel
210:
211: ' -----
212: objWMIInstance.Delete_
213: If Err.Number Then ErrorHandler (Err)
214:
215: WScript.Echo "PING timer deleted."
216:
217: Set objWMIServices = Nothing
218: Set objWMISink = Nothing
219:
220: WScript.Echo "Finished."
221:
...:
...:
...:
```

Once the WMI connection is established (lines 182 through 186), the script creates an interval timer instance (lines 190 through 198). During the command-line parameters parsing, the default interval is set to ten seconds if it is not specified. Next, the script submits the WQL event query (line 201) and enters an idle state while waiting for events (line 206).

When the script is stopped, the event subscription is canceled (lines 208 and 209), and the timer interval event is deleted (lines 212 and 213).

Sample 3.61 *PINGing a system at regular time intervals (Part II)*

```

...:
...:
...:
221:
222: ' -----
223: Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)
...:
229:     Wscript.Echo
230:     WScript.Echo FormatDateTime(Date, vbLongDate) & " at " & _
231:             FormatDateTime(Time, vbLongTime) & "." & vbCRLF & _
232:             " WMIPING host '" & strHost & "' started ..."
233:
234: ' -- PING -----
235: Set objWMIInstance = objWMIServices.Get _
236:             (cWMIPingStatusClass & ".Address=""" & strHost & """ & _
237:             "ResolveAddressNames=""" & boolResolveAddressNames & """ & _
238:             "BufferSize=""" & intBufferSize & """ & _
239:             "NoFragmentation=""" & boolNoFragmentation & """ & _
240:             "TimeToLive=""" & intTimeToLive & """ & _
241:             "TypeofService=""" & intTypeofService & """ & _
242:             "RecordRoute=""" & intRecordRoute & """ & _
```

```

243:             "TimestampRoute=" & intTimestampRoute & "," & _
244:             "SourceRoute=''' & strSourceRoute & '''," & _
245:             "SourceRouteType=" & intSourceRouteType & "," & _
246:             "Timeout=" & intTimeout)
...
249:     If objWMIInstance.PrimaryAddressResolutionStatus Then
250:         WScript.Echo Space (2) & _
251:             DecodeWinSockAPIErrors (objWMIInstance.PrimaryAddressResolutionStatus)
252:         If boolHasFailed = False And boolSendAlert Then
253:             SendAlert objWMIInstance, _
254:                 "WMIPing failed: " & _
255:                 "From " & objWMIInstance.SystemProperties_.Item("__SERVER") & " '" & _
256:                 DecodeWinSockAPIErrors(objWMIInstance.PrimaryAddressResolutionStatus) & _
257:                 & "' - " & FormatDateTime(Date, vbLongDate) & _
258:                 " at " & _
259:                 FormatDateTime(Time, vbLongTime)
260:             boolHasFailed = True
261:         End If
262:     Else
263:         If objWMIInstance.StatusCode Then
264:             WScript.Echo Space (2) & _
265:                 DecodePINGStatusCode (objWMIInstance.StatusCode)
266:             If boolHasFailed = False And boolSendAlert Then
267:                 SendAlert objWMIInstance, _
268:                     "WMIPing failed: " & _
269:                     "From " & objWMIInstance.SystemProperties_.Item("__SERVER") & " '" & _
270:                     DecodePINGStatusCode (objWMIInstance.StatusCode) & "' - " & _
271:                     FormatDateTime(Date, vbLongDate) & _
272:                     " at " & _
273:                     FormatDateTime(Time, vbLongTime)
274:                 boolHasFailed = True
275:             End If
276:         Else
277:             WScript.Echo " Host '" & strHost & "' successfully contacted."
278:             boolHasFailed = False
279:         End If
280:     End If
281:
282:     If boolVerbose Then
283:         DisplayFormattedProperties objWMIInstance, 2
284:         WScript.Echo
285:     End If
...
289:     WScript.Echo " WMIPING host '" & strHost & "' ended."
290:
291: End Sub
292:
293: ]]>
294: </script>
295: </job>
296:</package>
```

When the timer event notification occurs, the event sink routine is invoked (lines 223 through 291). Based on the parameters given on the command line (or the default parameters established during the command-line parsing operation), the script gets a *Win32_PingStatus* instance (lines 235 through 246). The properties of this instance reflect the results of the

PING command-line utility, which is displayed at line 283 if the verbose mode is requested.

If the PING naming resolution (line 249) or the PING status (line 263) failed, the script displays an error message (lines 250 and 251 or lines 264 and 265) and sends an email alert by reusing the SendAlert() function previously developed (lines 253 through 260 or lines 267 through 273). It is important to note that the script sends the email alert only one time. This is why a Boolean value is set to true once the alert has been sent (line 260 or 274). The next time the PING echo is successful, the Boolean value is reset to False (line 278).

3.7.2 Network Diagnostic provider

The *Network Diagnostic* provider exposes methods to test the IP connectivity to a particular host with eventually a specific port IP number. The provider is implemented as an instance and method provider, as shown in Table 3.59.

Table 3.59 The Network Diagnostic Providers 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
Network Diagnostic Provider																
NetDiagProv	Root/CIMV2	X	X					X	X	X	X		X			

This provider only supports one single class called *NetDiagnostics*, which is a singleton class. Available in the Windows XP release, this class has been removed from the Windows Server 2003 platform during the beta program. This class also exposes information about the proxy settings, such as the *Win32_Proxy* class. Since the *Win32_Proxy* class is especially designed to manage the proxy settings, it is the recommended class to work with. However, the *NetDiagnostics* class is quite interesting to use to test the TCP/IP port connectivity between two hosts. This is the purpose of Sample 3.62. This script sample is quite easy to use.

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

```
Usage: WMINetDiag.wsf [/PingAddress:value] [/PingPort:value]
                         [/Machine:value] [/User:value] [/Password:value]
```

Options:

```
PingAddress : IP or host name to connect to.
PingPort    : Port number to connect to.
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)
```

Examples:

```
WMINetDiag /PingAddress:proxy.LissWare.Net
WMINetDiag /PingAddress:proxy.LissWare.Net /PingPort:8080
```

Sample 3.62 shows how to script with this *NetDiagnostics* class.

→ **Sample 3.62** Testing connectivity with the Network Diagnostic provider

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
25:  </runtime>
26:
27:  <script language="VBScript" src=".\\Functions\\ReplaceStringFunction.vbs" />
28:  <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
29:
30:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
31:
32:  <script language="VBScript">
33:  <![CDATA[
.:
37:  Const cComputerName = "LocalHost"
38:  Const cWMINameSpace = "Root\\CIMv2"
39:  Const cWMIClass = "NetDiagnostics"
.:
59:  ' -----
60:  ' Parse the command line parameters
61:  If WScript.Arguments.Named.Count = 0 Then
62:      WScript.Arguments.ShowUsage()
63:      WScript.Quit
64:  End If
.:
89:  strComputerName = WScript.Arguments.Named("Machine")
90:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
91:
92:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
93:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
94:
95:  Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
96:                                         strUserID, strPassword)
.:
```

```

99:  Set objWMIInstance = objWMIServices.Get (cWMIClass & "=@")
...
102: If boolPing Then
103:   If intPingPort <> -1 Then
104:     boolRC = objWMIInstance.ConnectToPort (strPingAddress, intPingPort, strMessageOut)
105:     varTemp = "PINGING address '" & strPingAddress & ":" & intPingPort & "'"
106:   Else
107:     boolRC = objWMIInstance.Ping (strPingAddress, strMessageOut)
108:     varTemp = "PINGING address '" & strPingAddress & "'"
109:   End If
110:
111: If boolRC Then
112:   WScript.Echo varTemp & " is successful."
113: Else
114:   WScript.Echo varTemp & " has failed."
115: End If
116:
117: If Len (strMessageOut) Then
118:   ReplaceString strMessageOut, "<br>", vbCRLF
119:   WScript.Echo vbCRLF & "Message:" & vbCRLF & vbCRLF & strMessageOut
120: End If
121: End If
...
127:  ]]>
128:  </script>
129: </job>
130:</package>
```

Once the command line is parsed (lines 60 through 90) and the WMI connectivity is established (lines 92 through 96), the script retrieves the singleton instance of the *NetDiagnostics* class (line 99). If the /PingAddress switch is given, a ping to the given IP address is executed with the *Ping* method (lines 107 and 108). If the ping is successful, the following output message is returned:

```
C:\>WMINetDiag /PingAddress:proxy.LissWare.Net
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

PINGING address 'proxy.LissWare.Net' is successful.

Message:

pinging (proxy.LissWare.Net)
64 bytes from 10.10.10.254: icmp_seq = 0.  time: 380 ms
64 bytes from 10.10.10.254: icmp_seq = 1.  time: 421 ms
64 bytes from 10.10.10.254: icmp_seq = 2.  time: 400 ms
64 bytes from 10.10.10.254: icmp_seq = 3.  time: 401 ms
```

Basically, this method performs the same task as the *Win32_PingStatus* class. However, none of the parameters to ping a host exposed by the *Win32_PingStatus* class is available from the *NetDiagnostics* class, which makes the *Win32_PingStatus* class more suitable to use. The message returned from the *Ping* method (line 107) contains an HTML carriage

return
; the message is parsed at line 118 with the ReplaceString() function included at line 27. Every occurrence of the
 HTML tag will be replaced with a carriage return and a line feed (line 118).

If the /PingPort switch is given, the *ConnectToPort* method is executed (line 104). Instead of performing a simple ping, the method tries to establish a connection with the given host at the specified port number. The obtained output is the following:

```
C:\>WMINetDiag /PingAddress:proxy.LissWare.Net /PingPort:8080
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

PINGing address 'proxy.LissWare.Net:8080' is successful.
```

The method return code is tested to determine if there is an execution failure (line 111) and an appropriate message (built at line 105 or 108) is displayed (lines 112 through 114).

```
C:\>WMINetDiag /PingAddress:proxy.LissWare.Net /PingPort:8081
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

PINGing address 'proxy.LissWare.Net:8081' has failed.
```

3.7.3 IP routing provider

The *IP routing* providers give access to the IP version 4.0 routing table. They allow the examination and the modification of the IP routing table of a particular system. There are two *IP routing* providers: one instance provider, which gives access to the routing table information (called *RouteProvider*), and one event provider (called *RouteEventProvider*) able to trigger WMI events in case of modification of the routing table (see Table 3.60).

Table 3.60 The IP routing providers 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
IP Route Provider																
RouteEventProvider	Root/CIMV2		X									X	X			
RouteProvider	Root/CIMV2	X					X	X	X	X	X	X				

These providers are available in the `Root\CIMv2` namespace and support the classes listed in Table 3.61.

Table 3.61 *The IP routing Providers Classes*

Name	Type	Comments
Win32_IP4RouteTable	Dynamic	The <code>IP4RouteTable</code> class information governs where network data packets are routed (e.g., usually Internet packets are sent to a gateway, and local packets may be routed directly by the client's machine). Administrators can use this information to trace problems associated with misrouted packets, and also direct a computer to a new gateway as necessary. This class deals specifically with IP4 and does not address IPX or IP6. It is only intended to model the information revealed when typing the 'Route Print' command from the command prompt.
Win32_IP4PersistedRouteTable	Dynamic	The <code>IP4PersistedRouteTable</code> class contains IP routes that are persisted. By default, the routes you add to the routing table aren't permanent. You lose these routes when you reboot your computer. However, if you use the command <code>route -p add</code> , Windows NT makes them permanent, so you won't lose the route when you reboot your computer. Persistent entries are automatically reinserted in your route table each time your computer's route table is rebuilt. Windows NT stores persistent routes in the Registry. This class deals specifically with IP4 and does not address IPX or IP6.
Win32_ActiveRoute	Association	The <code>ActiveRoute</code> class associates the current IP4 Route being used with the persisted IP route table.
Win32_IP4RouteTableEvent	Event (Extrinsic)	The <code>Win32_IP4RouteTableEvent</code> class represents IP route change events resulting from the addition, removal, or modification of IP routes on the computer system.

Three classes work with the instance provider and one with the event provider. The `Win32_IP4RouteTableEvent` is an extrinsic class, which only provides timestamp information when an IP routing table modification occurs. To subscribe to this type of event, the following WQL event query must be used:

```
Select * From Win32_IP4RouteTableEvent
```

Once executed, the event returns the following information:

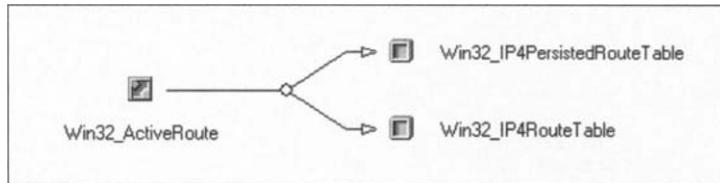
```

1: C:\GenericEventAsyncConsumer.wsf "Select * From Win32_IP4RouteTableEvent"
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Saturday, 15 December, 2001 at 13:41:29: 'Win32_IP4RouteTableEvent' has been triggered.
9:
10: - Win32_IP4RouteTableEvent -----
11:   TIME_CREATED: ..... 15-12-2001 12:41:29 (20011215124129.409259+060)
12:
13: END - OnObjectReady.
```

The routing table is a collection of instances from the `Win32_IP4RouteTable` class. However, each time a route is a persistent route, an association exists with the `Win32_IP4PersistedRouteTable` class. The associa-

tion class used to link these two classes together is the *Win32_ActiveRoute* class (see Figure 3.35).

Figure 3.35
The *Win32_IP4RouteTable* association.



We will take advantage of this association to determine if a route is a persistent route or to create a new persistent route. The purpose of Samples 3.63 through 3.65 is to illustrate the scripting logic to use to view, add, and delete IP routes (persistent routes or not).

The command-line parameters exposed by the script are as follows:

```

C:\>WMIIP4Route.wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Usage: WMIIP4Route.wsf Action Destination MASK Mask NextHop [METRIC Metric] [IF IfNumber]
      [/Persistent[+|-]] [/Machine:value] [/User:value] [/Password:value]

Options:

Action      : Specify the operation to perform: [Print] or [Add] or [Delete].
Destination : IP network destination.
MASK Mask   : Mask used by the specified IP network.
NextHop     : IP address of the gateway to access the specified IP network.
METRIC Metric : Specify the metric used.
IF IfNumber : Specify the Network Interface Number to use.
Persistent  : Make the specified route persistent.
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)

Example:

WMIIP4Route.wsf PRINT
WMIIP4Route.wsf ADD 205.10.10.0 MASK 255.255.255.0 10.10.10.253 METRIC 30 IF 2
WMIIP4Route.wsf ADD 205.10.10.0 MASK 255.255.255.0 10.10.10.253 METRIC 30 IF 2 /Persistent+
WMIIP4Route.wsf DELETE 204.10.10.0 MASK 255.255.255.0 10.10.10.253
  
```

First, you will note that the script accepts commands that are similar to the **ROUTE.Exe** command-line utility. This will make the user more familiar with the script usage. Before diving into the code, let's see what the output of the script **PRINT** command looks like.

```
C:\>WMIIIP4Route PRINT
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
```

ACTIVE ROUTES =====

Destination	Mask	NextHop	Metric	IfNumber	Protocol	Type	Persistent
0.0.0.0	0.0.0.0	10.10.10.254	30	2	Netmgmt	Indirect	-
10.0.0.0	255.0.0.0	10.10.10.3	30	2	Local	Direct	-
10.10.10.3	255.255.255.255	127.0.0.1	30	1	Local	Direct	-
10.255.255.255	255.255.255.255	10.10.10.3	30	2	Local	Direct	-
127.0.0.0	255.0.0.0	127.0.0.1	1	1	Local	Direct	-
192.10.10.0	255.255.255.0	10.10.10.253	1	2	10010	Indirect	Y
193.10.10.0	255.255.255.0	10.10.10.253	1	2	10010	Indirect	Y
194.10.10.0	255.255.255.0	10.10.10.253	1	2	10010	Indirect	Y
195.10.10.0	255.255.255.0	10.10.10.253	1	2	10010	Indirect	Y
200.10.10.0	255.255.255.0	10.10.10.253	1	2	10010	Indirect	Y
205.10.10.0	255.255.255.0	10.10.10.253	30	2	Netmgmt	Indirect	Y
224.0.0.0	240.0.0.0	10.10.10.3	30	2	Local	Direct	-
255.255.255.255	255.255.255.255	10.10.10.3	1	2	Local	Direct	-

Completed.

As you can see, the output is quite similar to the one obtained with the ROUTE.Exe PRINT command. Let's see how the code producing this output works by looking at the first part of the script shown in Sample 3.63. As usual, the lines defining and parsing the command-line parameters are skipped.

Sample 3.63 Viewing, adding, and deleting IP v4.0 routes (Part I)

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <runtime>
.:
32:   </runtime>
33:
34:
35:   <script language="VBScript" src=".\\Functions\\DecodeIPRouteFunction.vbs" />
36:   <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
37:
38:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
39:   <object progid="Wscript.Network" id="WshNetwork" reference="true"/>
40:
41:   <script language="VBScript">
42:     <![CDATA[
.:
46:     Const cComputerName = "localhost"
47:     Const cWMINNameSpace = "Root/cimv2"
48:     Const cWMIIIP4RouteClass = "Win32_IP4RouteTable"
49:     Const cWMIIIPersistedRouteClass = "Win32_IP4PersistedRouteTable"
50:     Const cWMIAactiveRouteClass = "Win32_ActiveRoute"
```

```
51:      Const cNotFound = &h80041002
...
159:      strComputerName = WScript.Arguments.Named("Machine")
160:      If Len(strComputerName) = 0 Then strComputerName = cComputerName
161:
162:      objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
163:      objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
164:
165:      Set objWMIServices = objWMIConnector.ConnectServer(strComputerName, cWMINamespace, _
166:                                         strUserID, strPassword)
...
169:      ' -- PRINT -----
170:      If boolPrint = True Then
171:          Set objWMIP4RouteInstances = objWMIServices.InstancesOf (cWMIP4RouteClass)
172:
173:          WScript.Echo "ACTIVE ROUTES " & String (84, "=") & vbCrLf
174:          WScript.Echo "    Destination           Mask           NextHop" & _
175:                      " Metric  IfNumber  Protocol      Type  Persistent"
176:          WScript.Echo String (98, "-")
177:
178:          For Each objWMIP4RouteInstance In objWMIP4RouteInstances
179:              strIPProtocol = DecodeIPProtocolCode (objWMIP4RouteInstance.Protocol)
180:              strIP4RouteType = DecodeIPRouteType (objWMIP4RouteInstance.Type)
181:
182:              Set objWMIPersistentInstances = objWMIServices.ExecQuery _
183:                                         ("Associators of {" & _
184:                                         "                    objWMIP4RouteInstance.Path_.RelPath & _
185:                                         "} Where AssocClass=Win32_ActiveRoute")
...
188:              If objWMIPersistentInstances.Count = 1 Then
189:                  strPersistent = "        Y"
190:              Else
191:                  strPersistent = "        -"
192:              End If
...
196:              WScript.Echo String (15 - Len (objWMIP4RouteInstance.Destination), " ") & _
197:                           objWMIP4RouteInstance.Destination & " " & _
198:                           String (15 - Len (objWMIP4RouteInstance.Mask), " ") & _
199:                           objWMIP4RouteInstance.Mask & _
200:                           String (15 - Len (objWMIP4RouteInstance.NextHop), " ") & _
201:                           objWMIP4RouteInstance.NextHop & _
202:                           String (10 - Len (objWMIP4RouteInstance.Metric1), " ") & _
203:                           objWMIP4RouteInstance.Metric1 & _
204:                           String (10 - Len (objWMIP4RouteInstance.InterfaceIndex), " ") & _
205:                           objWMIP4RouteInstance.InterfaceIndex & _
206:                           String (10 - Len (strIPProtocol), " ") & _
207:                           strIPProtocol & _
208:                           String (10 - Len (strIP4RouteType), " ") & _
209:                           strIP4RouteType & "  " & _
210:                           strPersistent
211:              Next
212:
213:              WScript.Echo vbCrLf & "Completed."
...
216:      End If
...
...
...
```

To print the routing table information, the basic principle is quite easy, since it consists of the retrieval of all instances available from the *Win32_IP4RouteTable* class (line 171). Next, the script enumerates in a loop all instances available (lines 178 through 211). Before displaying the properties of each IP route instance, the script checks if an association exists with an instance of the *Win32_IP4PersistedRouteTable* class (lines 182 through 192). If this is the case, it means that the examined IP route instance is a persistent route (line 189). The output is constructed in the loop (lines 196 through 210). Because some properties of *Win32_IP4RouteTable* class contain numbers having a particular meaning, the script includes the *DecodeIPRouteFunction.vbs* file (line 35), which contains several functions to decode the value of some properties (lines 179 and 180).

The *Win32_IP4RouteTable* class does not expose any method to add a new IP route. The creation of a new IP route is made by the creation of a new *Win32_IP4RouteTable* instance, as shown in Sample 3.64. However, the creation of the new instance is only made if there is no existing instance of the IP route to be added. This logic is implemented by trying first to retrieve an existing instance of the IP route to be added (lines 220 through 223). If the route exists, it will be updated with the new parameters. If the route does not exist (line 223), a new IP route will be created (lines 225 and 226). Next, independently of a route addition or update, the route parameters are set from line 234 through 240. Then the route is created or updated by using the *Put_* method of the *SWBemObject* representing the IP route instance (lines 242 and 243).

Sample 3.64 *Adding IP v4.0 routes (Part II)*

```
...:  
...:  
...:  
217:  
218: ' -- Add -----  
219: If boolAdd = True Then  
220:   Set objWMIIP4RouteInstance = objWMIServices.Get _  
221:     (cWMIIP4RouteClass & ".Destination=''" & strDestination & _  
222:      "','" & strNextHop & "")  
223: If Err.Number = cNotFound Then  
224:   Err.Clear  
225:   Set objWMIClass = objWMIServices.Get (cWMIIP4RouteClass)  
226:   Set objWMIIP4RouteInstance = objWMIClass.SpawnInstance_  
227:   If Err.Number Then  
228:     ErrorHandler (Err)  
229:   End If  
...:  
232: End If  
233:  
234: objWMIIP4RouteInstance.Destination = strDestination  
235: objWMIIP4RouteInstance.Mask = strMask  
236: objWMIIP4RouteInstance.NextHop = strNextHop
```

```
237:     objWMIP4RouteInstance.Metric1 = intMetric
238:     objWMIP4RouteInstance.InterfaceIndex = intIndex
239:     objWMIP4RouteInstance.Protocol = intProtocol
240:     objWMIP4RouteInstance.Type = intRouteType
241:
242:     objWMIP4RouteInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
243:                                 wbemFlagReturnWhenComplete)
...
246: If boolPersistent Then
247:     Set objWMIPersistentInstance = objWMIInstances.Get _
248:         (cWMIPersistentRouteClass & _
249:             ".Destination=""" & strDestination & _
250:             ",Mask=""" & strMask & _
251:             ",Metric1=""" & intMetric & _
252:             ",NextHop=""" & strNextHop & """)
253: If Err.Number = cNotFound Then
254:     Err.Clear
255:     Set objWMIClass = objWMIInstances.Get (cWMIPersistentRouteClass)
256:     Set objWMIPersistentInstance = objWMIClass.SpawnInstance_
...
262: End If
263:
264:     objWMIPersistentInstance.Destination = strDestination
265:     objWMIPersistentInstance.Mask = strMask
266:     objWMIPersistentInstance.NextHop = strNextHop
267:     objWMIPersistentInstance.Metric1 = intMetric
268:
269:     objWMIPersistentInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
270:                                   wbemFlagReturnWhenComplete)
...
273: Set objWMIAssocInstance = objWMIInstances.Get _
274:     (cWMIActiveRouteClass & ".SameElement=""" & _
275:         objWMIPersistentInstance.Path_.Path & _
276:         ",SystemElement=""" & _
277:             objWMIP4RouteInstance.Path_.Path & """)
278: If Err.Number = cNotFound Then
279:     Err.Clear
280:     Set objWMIClass = objWMIInstances.Get (cWMIPersistentRouteClass)
281:     Set objWMIAssocInstance = objWMIClass.SpawnInstance_
...
288:     objWMIAssocInstance.SameElement = objWMIPersistentInstance.Path_.Path
289:     objWMIAssocInstance.SystemElement = objWMIP4RouteInstance.Path_.Path
290:
291:     objWMIAssocInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
292:                               wbemFlagReturnWhenComplete)
...
295: End If
...
300: WScript.Echo "IP route " & strDestination & _
301:     " MASK " & strMask & " " & _
302:     strNextHop & " added as a persistent route."
303:
304: Else
305:     WScript.Echo "IP route " & strDestination & _
306:         " MASK " & strMask & " " & _
307:             strNextHop & " added."
308: End If
...
311: End If
312:
...
...
...
```

Next, if the */Persistent+* switch is specified on the command line, the script creates the association that must be in place to make the route persistent (lines 247 through 303). Along these lines two instances are created, as follows:

- One instance is made from the *Win32_IP4PersistedRouteTable* class (lines 247 through 270): The creation of this instance follows logic similar to the *Win32_IP4RouteTable* instance. The script checks first if no instance exists by trying to retrieve the instance (lines 247 through 252); if this operation fails (line 253), then a new instance is created (lines 255 and 256). Next, the instance properties are set (lines 264 through 267) and the information is saved in the CIM repository (lines 269 and 270).
- One association instance is made from the *Win32_ActiveRoute* class (lines 273 through 292): Here, again, the instance logic creation is the same. The script verifies first if a *Win32_ActiveRoute* instance exists (lines 273 through 277) and if not, a new *Win32_ActiveRoute* instance is created (lines 280 and 281). To create or update the *Win32_ActiveRoute* association instance, the script reuses the WMI path of the two previous instances: the *Win32_IP4RouteTable* instance path (line 289) and the *Win32_IP4PersistedRouteTable* instance (line 288). Next, the information is committed into the CIM repository (lines 291 and 292).

The deletion of a *Win32_IP4RouteTable* instance is much easier. The logic is shown in Sample 3.65.

Sample 3.65

Deleting IP v4.0 routes (Part III)

```
...:
...:
...:
312:
313: ' -- DELETE -----
314: If boolDelete = True Then
315:     Set objWMIP4RouteInstance = objWMIservices.Get _
316:         (cWMIP4RouteClass & ".Destination=''' & strDestination & _
317:             ''',NextHop=''' & strNextHop & '''")
...:
320:     Set objWMIPersistentInstances = objWMIservices.ExecQuery _
321:                     ("Associators of {" & _
322:                         objWMIP4RouteInstance.Path_.RelPath & _
323:                             "} Where AssocClass=Win32_ActiveRoute")
...:
326: If objWMIPersistentInstances.Count = 1 Then
327:     Set objWMIPersistentInstance = objWMIservices.Get _
```

```
328:          (cWMIIPPersistedRouteClass & _
329:           ".Destination="" & objWMIIIP4RouteInstance.Destination & _
330:           "' ,Mask=""" & objWMIIIP4RouteInstance.Mask & _
331:           "' ,Metric1=""" & objWMIIIP4RouteInstance.Metric1 & _
332:           "' ,NextHop=""" & objWMIIIP4RouteInstance.NextHop & ""))
...
335:      objWMIPersistentInstance.Delete_
...
339:  End If
...
343:  objWMIIIP4RouteInstance.Delete_
...
346:  WScript.Echo "IP route " & strDestination & _
347:           " MASK " & strMask & " " & -
348:           strNextHop & " deleted."
...
351:  End If
...
355:  ]]>
356: </script>
357: </job>
358:</package>
```

First, the script retrieves the *Win32_IP4RouteTable* instance to delete (lines 315 through 317). Next, before executing the deletion of the retrieved instance, it is important to verify if there is an association with a *Win32_IP4PersistedRouteTable* instance. This verification is performed from line 320 through 326. If there is an association, the *Win32_IP4PersistedRouteTable* instance must also be retrieved (lines 327 through 332). Once all instances are retrieved, the script deletes each of them (lines 335 and 343). It is important to delete the associated instance, because, if a similar route is recreated, this route will be considered as a persistent route even if the */Persistent+* switch is not specified. The simple existence of a *Win32_IP4PersistedRouteTable* instance matching the *Win32_IP4RouteTable* information will make the route persistent.

3.7.4 DNS provider

The *DNS* provider supports the management of the DNS server, the DNS zones, and the DNS records. It is made up of one provider, which is both an instance provider and a method provider. Only available under Windows Server 2003 by default, the *DNS* provider was also available for Windows 2000 at <ftp://ftp.microsoft.com/reskit/win2000/dnsprov.zip>. With this provider it is possible to retrieve (“Get” and “Enumeration”), modify (“Put”), and delete (“Delete”) DNS information. The provider capabilities are summarized in Table 3.62.

Table 3.62 The DNS Providers 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
DNS Provider	Root\MicrosoftDNS	X	X			X	X	X	X	X				X		
MS_NT_DNS_PROVIDER	Root\MicrosoftDNS															

The *DNS* provider is available in the `Root\MicrosoftDNS` WMI namespace. It supports more than 30 classes, where most of them represent the various DNS records. Each DNS manageable component is represented with a corresponding WMI class, as shown in Table 3.63.

Table 3.63 The DNS Providers Classes

Name	Type	Comments
MicrosoftDNS_Cache	Dynamic	This class describes a cache existing on a DNS server. It shouldn't be confused with a Cache file which contains root hints. This class simplifies visualizing the containment of DNS objects, rather than representing a real object. The class, <code>MicrosoftDNS_Cache</code> , is a container for the resource records cached by the DNS server. Every instance of the class <code>MicrosoftDNS_Cache</code> must be assigned to one and only one DNS server. It may be associated with (or more intuitively 'may contain') any number of instances of the classes, <code>MicrosoftDNS_Domain</code> and/or <code>MicrosoftDNS_ResourceRecord</code> .
MicrosoftDNS_Domain	Dynamic	This class represents a Domain in a DNS hierarchy tree.
MicrosoftDNS_ResourceRecord	Dynamic	This class represents the general properties of a DNS Resource Record.
MicrosoftDNS_RootHints	Dynamic	This class describes the Root Hints stored in a Cache file on a DNS server. This class simplifies visualizing the containment of DNS objects, rather than representing a real object. Class <code>MicrosoftDNS_RootHints</code> is a container for the resource records stored by the DNS server in a Cache file. Every instance of the class <code>MicrosoftDNS_RootHints</code> must be assigned to one and only one DNS server. It may be associated with (or more intuitively 'may contain') any number of instances of class <code>MicrosoftDNS_ResourceRecord</code> .
MicrosoftDNS_Server	Dynamic	This class describes a DNS server. Every instance of this class may be associated with (or more intuitively 'may contain') one instance of class <code>MicrosoftDNS_Cache</code> , one instance of class <code>MicrosoftDNS_RootHints</code> and multiple instances of class <code>MicrosoftDNS_Zone</code> .
MicrosoftDNS_Statistic	Dynamic	A single DNS Server statistic.
MicrosoftDNS_Zone	Dynamic	This class describes a DNS Zone. Every instance of the class <code>MicrosoftDNS_Zone</code> must be assigned to one and only one DNS server. Zones may be associated with (or more intuitively 'may contain') any number of instances of the classes <code>MicrosoftDNS_Domain</code> and/or <code>MicrosoftDNS_ResourceRecord</code> .
MicrosoftDNS_DomainDomainContainment	Association	Domains may contain other Domains. (Every instance of the <code>MicrosoftDNS_Domain</code> class may contain multiple other instances of <code>MicrosoftDNS_Domain</code> .) An instance of a <code>MicrosoftDNS_Domain</code> object is directly contained in (at most) one higher-level <code>MicrosoftDNS_Domain</code> .
MicrosoftDNS_DomainResourceRecordContainment	Association	Every instance of the class <code>MicrosoftDNS_Domain</code> may contain multiple instances of the class, <code>MicrosoftDNS_ResourceRecord</code> . Every instance of the class <code>MicrosoftDNS_ResourceRecord</code> belongs to a single instance of the class <code>MicrosoftDNS_Domain</code> and is defined to be weak to that instance.
MicrosoftDNS_ServerDomainContainment	Association	Every instance of the class <code>MicrosoftDNS_Server</code> may contain multiple instances of the class <code>MicrosoftDNS_Domain</code> . Every instance of the class <code>MicrosoftDNS_Domain</code> belongs to a single instance of the class <code>MicrosoftDNS_Server</code> and is defined to be weak to that server.

Table 3.63 *The DNS Providers Classes (continued)*

Name	Type	Comments
MicrosoftDNS_AAAAType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type AAAA record.
MicrosoftDNS_AFSDBType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type AFSDB record.
MicrosoftDNS_ATMAType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type ATMA record.
MicrosoftDNS_AType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type A record.
MicrosoftDNS_CNAMEType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type CNAME record.
MicrosoftDNS_HINFOType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type HINFO record.
MicrosoftDNS_ISDNType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type ISDN record.
MicrosoftDNS_KEYType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type KEY record.
MicrosoftDNS_MBType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MB record.
MicrosoftDNS_MDTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MD record.
MicrosoftDNS_MFTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MF record.
MicrosoftDNS_MGType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MG record.
MicrosoftDNS_MINFOType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MINFO record.
MicrosoftDNS_MRType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MR record.
MicrosoftDNS_MXType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type MX record.
MicrosoftDNS_NSType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type NS record.
MicrosoftDNS_NXTType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type NXT record.
MicrosoftDNS_PTRType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type PTR record.
MicrosoftDNS_RPTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type RPT record.
MicrosoftDNS_RTTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type RTT record.
MicrosoftDNS_SIGType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type SIG record.
MicrosoftDNS_SOATypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type SOA record.
MicrosoftDNS_SRVType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type SRV record.
MicrosoftDNS_TXTType	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type TXT record.
MicrosoftDNS_WINSRTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type WINSR record.
MicrosoftDNS_WINSTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type WINS C12 record.
MicrosoftDNS_WKSTypE	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type WKS record.
MicrosoftDNS_X25Type	Dynamic	A subclass of MicrosoftDNS_ResourceRecord that represents a Type X25 record.

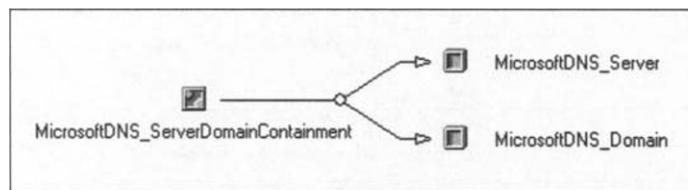
It is important to note that the *DNS* provider is not implemented as a WMI event provider, which forces you to use the **WITHIN** statement in a WQL event query. For example, to monitor the A record modifications made in the LissWare.Net domain, the following WQL event query would be used:

```
Select * From __InstanceCreationEvent Within 10 Where TargetInstance ISA 'MicrosoftDNS_AType'
And TargetInstance.DomainName='LissWare.Net'
```

When modifying an existing record, it is important to note that no `_InstanceModificationEvent` intrinsic event is returned. A DNS record update corresponds to a DNS record deletion followed by a DNS record creation. This is the reason why the WQL event query sample uses the `_InstanceCreationEvent` intrinsic event class instead of the `_InstanceModificationEvent` intrinsic event class.

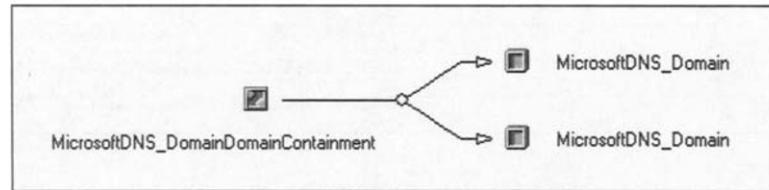
Because there is no direct relationship between the DNS server host name and the domain definitions that it could contain, the DNS CIM representation defines an association class called `MicrosoftDNS_ServerDomainContainment` to associate the `MicrosoftDNS_Server` and the `MicrosoftDNS_Domain` classes, as shown in Figure 3.36.

Figure 3.36
The DNS server class is associated with the DNS domain class.



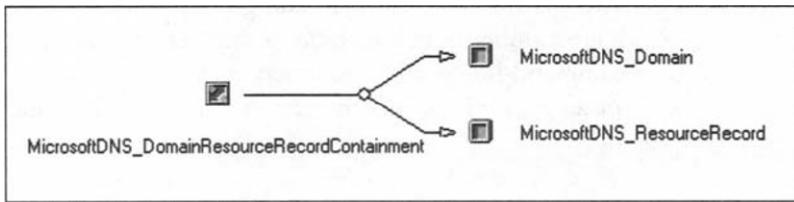
In the same way, because a DNS domain may contain another DNS domain, an association class called `MicrosoftDNS_DomainDomainContainment` establishes the relationship with the `MicrosoftDNS_Domain` class itself, as shown in Figure 3.37. The `MicrosoftDNS_Domain` is a superclass used for the definition of three subclasses called the `MicrosoftDNS_RootHints` class, the `MicrosoftDNS_Cache` class, and the `MicrosoftDNS_Zone`. It is interesting to note that a DNS domain delegation will be represented as an association of two `MicrosoftDNS_Domain` instances.

Figure 3.37
The DNS domain class is associated with itself.



Last but not least, because a DNS domain usually contains DNS records, an association class called `MicrosoftDNS_DomainResourceRecordContainment` establishes the relationship between the `MicrosoftDNS_Domain` superclass and the `MicrosoftDNS_ResourceRecord` superclass, as shown in Figure 3.38.

Figure 3.38
The DNS domain class is associated with the DNS records superclass.



The *MicrosoftDNS_ResourceRecord* class is used as a superclass because it is used as a parent class for all record-specific classes listed in Table 3.63.

These associations can be exploited to retrieve items contained in various DNS containers. For example, to retrieve the list of domains hosted in a DNS server called NET-DPEN6400A in the LissWare.Net domain, we will use the following WQL data query:

```
Associators of {MicrosoftDNS_Server.Name="net-dpen6400a.LissWare.Net"} Where
  AssocClass=MicrosoftDNS_ServerDomainContainment
```

The *MicrosoftDNS_Server* class is using only one key property, which is the Fully Qualified Domain Name (FQDN) of the DNS server. Thus, the WQL query only requires the name of the server for the selection.

In the same way, to retrieve the list of records that exist in a zone called LissWare.Net hosted in a server called NET-DPEN6400A, which is part of the LissWare.Net domain, the following WQL data query will be used:

```
Associators of {MicrosoftDNS_Zone.ContainerName="LissWare.Net",
  DnsServerName="net-dpen6400a.LissWare.Net",
  Name="LissWare.Net"} Where
  AssocClass=MicrosoftDNS_DomainResourceRecordContainment
```

Because the *MicrosoftDNS_Zone* has three key properties, the WQL query specifies the three key properties for the selection. This makes the WQL query quite complex to formulate in order to retrieve the DNS record list for a specific zone. Because the *MicrosoftDNS_ResourceRecord* class exposes a property that contains the container name, which is the domain name, it is possible to use the following WQL query instead:

```
Select * From MicrosoftDNS_ResourceRecord Where ContainerName='LissWare.Net'
```

Of course, in this case we do not exploit the associations in place. However, we use the fact that the *MicrosoftDNS_ResourceRecord* class is a superclass to simplify the WQL query.

The next script sample, shown in Samples 3.66 through 3.69, utilizes the *DNS* provider capabilities to manage the DNS server, the DNS zones, and DNS records from the command-line. It also exploits some of the char-

acteristics we have seen regarding the DNS association classes. Due to the huge number of configurable parameters exposed by the DNS classes, the command-line parameters syntax is especially adapted. For example, to create an Active Directory integrated zone, the command would be as follows:

```
C:\>WMIDNS.Wsf Zone Create Name=LissWare.Net ZType=ADIntegrated
AdminEmailName=Administrator@LissWare.Net
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Zone 'LissWare.Net' created.
```

Once the zone is created, it is possible to create some records—for example, an A record. In this case, the command line would be as follows:

```
C:\>WMIDNS.Wsf Record Create RType=A DomainName=LissWare.Net Host=Net-dopen6400a
Class=IN TTL=3600 HostAddress=10.10.10.7
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

'A' record for 'Net-dopen6400a.LissWare.Net' created.
```

And to create a DNS alias for the previously created A record, the command would be as follows:

```
C:\>WMIDNS.Wsf Record Create RType=CNAME DomainName=LissWare.Net Host=www
Class=IN TTL=7200 HostAddress=Net-dopen6400a.LissWare.Net
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

'CNAME' record for 'www.LissWare.Net' created.
```

The following is a complete list of command-line parameters supported by the next sample script.

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

Usage: WMIDNS.wsf DNS component [SERVER View] [SERVER Start] [SERVER Stop] [SERVER Scavenging]
       [ZONE View] [ZONE Create] [ZONE Update] [ZONE AgeAllRecords]
       [ZONE Refresh] [ZONE Pause] [ZONE Reload]
```

Options:

DNS component	: Select the DNS item to customize. Can be [Server], [Zone], [Cache], [RootHints], [Record]
SERVER View	: Views the DNS Server properties.
SERVER Start	: Starts the DNS Service.
SERVER Stop	: Stops the DNS Service.
SERVER Scavenging	: Scavenging of stale records in the zones subjected to scavenging.
SERVER BootMethod	: Defines the boot method. Only [File], [Registry] or [AD] is accepted.
SERVER EventLogLevel	: Defines the logging level. Only [None], [Errors], [ErrorsAndWarnings] or [All] is accepted.
SERVER Forwarders	: Defines the IP address list used as forwarders.
SERVER IsSlave	: Defines if the DNS server acts as a slave server. Only [True] or [False] is accepted.

```

SERVER NoRecursion      : Defines the recursion usage. Only [Disable] or [Enable] is accepted.
SERVER RoundRobin        : Define the round robin usage. Only [True] or [False] is accepted.
SERVER ListenAddresses   : Define the IP address used to listen DNS requests.
ZONE View                : Views the Zone properties.
ZONE Create              : Creates a DNS new zone.
ZONE Update              : Updates an existing DNS zone.
ZONE AgeAllRecords       : Enables aging for some or all non-NS and non-SOA records in a zone.
ZONE Refresh             : Forces the DNS server to check the master server of a secondary
                           zone for updates.
ZONE Pause               : Pauses the zone.
ZONE Reload              : Reloads the zone.
ZONE ResetSecondaries    : Resets the secondary IP addresses.
ZONE Resume              : Resumes the zone.
ZONE UpdateFromDS        : Forces an update of the zone from the DS. This method is
                           only valid for DS-integrated zones.
ZONE WriteBackZone       : Saves the zone's data to persistent storage.
ZONE Delete              : Deletes a zone
  Name                  : Name of the zone to manage.
  ZType                 : Zone type. Can be [ADIntegrated], [Primary], [Secondary] or [Stub]
  DataFile              : Zone file containing the zone data.
  AdminEmailName        : Zone administrator email address.
  IPAddresses           : Secondary DNS servers IP addresses.
  SecondaryServers      : IP addresses of the secondary servers.
  SecureSecondaries     : Secondary DNS zone transfer mode. Only [AnyServers], [OnlyNSServers],
                           [OnlyListedServers] or [None] is accepted.
  NotifyServers         : IP addresses of the secondary servers to notify.
  Notify                : Notify mode. Only [OnlyNSServers], [OnlyListedServers]
                           or [None] is accepted.
CACHE Clear              : Clears the DNS cache.
ROOTHINTS WriteBackDataFile : Saves the RootHints' zone data to persistent storage.
RECORD View               : View DNS records for a given zone.
RECORD Create             : Create DNS records in a given zone.
RECORD Delete             : Delete DNS records from a given zone.
  RType                 : Record Type. Can be [A], [CNAME], [NS], ... or any
                           record type supported.
  DomainName            : Record's domain name
  Host                  : Corresponding host to the record.
  Class                 : Class of the record.
  TTL                   : Time to Live of the record.
  HostAddress           : HostAddress of the record (Could be an IP address or a host
                           name based on the record type).
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)

```

Examples:

```

WMIDNS.Wsf Server View
WMIDNS.Wsf Server Start
WMIDNS.Wsf Server Stop
WMIDNS.Wsf Server Scavenging
WMIDNS.Wsf Server BootMethod=Registry
WMIDNS.Wsf Server EventLogLevel=ErrorsAndWarnings
WMIDNS.Wsf Server Forwarders=10.10.10.1,172.16.23.1
WMIDNS.Wsf Server IsSlave=True
WMIDNS.Wsf Server NoRecursion=Disabled
WMIDNS.Wsf Server RoundRobin=True
WMIDNS.Wsf Server ListenAddresses=10.10.10.3,10.10.10.4
WMIDNS.Wsf Server ListenAddresses=10.10.10.3 RoundRobin=True Forwarders=10.10.10.1
                           BootMethod=Registry EventLogLevel=ErrorsAndWarnings

```

```

WMIDNS.Wsf Zone View
WMIDNS.Wsf Zone View Name=LissWare.Net
WMIDNS.Wsf Zone Create Name=LissWare.Net ZType=ADIntegrated
    AdminEmailName=Administrator@LissWare.Net
WMIDNS.Wsf Zone Create Name=LissWare.Net ZType=Primary DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net
WMIDNS.Wsf Zone Create Name=LissWare.Net ZType=Secondary DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net IPAddresses=10.10.10.3
WMIDNS.Wsf Zone Create Name=LissWare.Net ZType=Stub DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net IPAddresses=10.10.10.3
WMIDNS.Wsf Zone Update Name=LissWare.Net ZType=ADIntegrated
    AdminEmailName=Administrator@LissWare.Net
WMIDNS.Wsf Zone Update Name=LissWare.Net ZType=Primary DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net
WMIDNS.Wsf Zone Update Name=LissWare.Net ZType=Primary DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net ZUpdate=NotAllowed
WMIDNS.Wsf Zone Update Name=LissWare.Net ZType=Secondary DataFile=LissWare.Net.Dns
    AdminEmailName=Administrator@LissWare.Net IPAddresses=10.10.10.3
WMIDNS.Wsf Zone Update Name=LissWare.Net ZUpdate=NotAllowed
WMIDNS.Wsf Zone Update Name=LissWare.Net ZUpdate=AllowUpdates
WMIDNS.Wsf Zone Update Name=LissWare.Net ZUpdate=AllowSecureUpdates
WMIDNS.Wsf Zone Refresh Name=LissWare.Net
WMIDNS.Wsf Zone Pause Name=LissWare.Net
WMIDNS.Wsf Zone Reload Name=LissWare.Net
WMIDNS.Wsf Zone ResetSecondaries Name=LissWare.Net SecondaryServers=10.10.10.4,10.10.10.5
    SecureSecondaries=OnlyListedServers
    NotifyServers=10.10.10.4,10.10.10.5 Notify=OnlyListedServers
WMIDNS.Wsf Zone Resume Name=LissWare.Net
WMIDNS.Wsf Zone UpdateFromDS Name=LissWare.Net
WMIDNS.Wsf Zone WriteBackZone Name=LissWare.Net
WMIDNS.Wsf Zone Delete Name=LissWare.Net

WMIDNS.Wsf Cache Clear
WMIDNS.Wsf RootHints WriteBackDataFile

WMIDNS.Wsf Record View RType=A DomainName=LissWare.Net
WMIDNS.Wsf Record Create RType=A DomainName=LissWare.Net Host=Net-dopen6400a Class=IN
    TTL=3600 HostAddress=10.10.10.7
WMIDNS.Wsf Record Create RType=CNAME DomainName=LissWare.Net Host=www Class=IN
    TTL=7200 HostAddress=Net-dopen6400a.LissWare.Net
WMIDNS.Wsf Record Delete RType=A DomainName=LissWare.Net Host=Net-dopen6400a Class=IN
    TTL=3600 HostAddress=10.10.10.7
WMIDNS.Wsf Record Delete RType=CNAME DomainName=LissWare.Net Host=www Class=IN
    TTL=7200 HostAddress=Net-dopen6400a.LissWare.Net.

```

Note that Samples 3.66 through 3.69 make use of the most important parameters exposed by the DNS classes. To simplify the script coding and focus on the DNS class usage, the script performs a very limited syntax checking. However, if you require a stronger syntax checking and some properties supported by the DNS classes that are not currently implemented in Samples 3.66 through 3.69, the script is very easy to extend, since its logic is quite linear. Although a smarter scripting technique is possible to make the script smaller, this linear logic has the advantage of being easy to follow and understand, which is the primary purpose.

We can split Samples 3.66 through 3.69 into the following three parts:

- A portion to manage the DNS server, shown in Sample 3.66
- A portion to manage the DNS zones, shown in Samples 3.67 and 3.68
- A portion to manage the DNS records, shown in Sample 3.69

Once the WMI connection is established in the `Root\MicrosoftDNS` namespace (lines 569 through 573), the script retrieves the name of the DNS server available in the system (lines 576 through 582). Normally, we only have one DNS server per system, which means that we may expect the `MicrosoftDNS_Server` class to be defined as a singleton class. However, the class definition is not made that way, since it uses a key property called `name`, which contains the FQDN of the DNS server (line 580). The FQDN of the server is retrieved at the beginning of the script, simply because its `name` will be used to manage the server, the zones, and the records.

The rest of the script makes use of some of the `MicrosoftDNS_Server` methods and properties (lines 587 through 757).

Sample 3.66*Managing the DNS server (Part I)*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <runtime>
.:
108: </runtime>
109:
110:  <script language="VBScript" src=".\\Functions\\DecodeDNSFunction.vbs" />
111:  <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
112:
113:  <script language="VBScript" src=".\\Functions\\ConvertStringInArrayFunction.vbs" />
114:  <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
115:
116:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
117:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
118:
119:  <script language="VBScript">
120:  <![CDATA[
.:
124:  Const cComputerName = "LocalHost"
125:  Const cWMINameSpace = "Root/MicrosoftDNS"
.:
569:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
570:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
571:
572:  Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
```

```
573:                                     strUserID, strPassword)
...
576:     Set objWMIIInstances = objWMIServices.InstancesOf ("MicrosoftDNS_Server")
577:
578:     If objWMIIInstances.Count = 1 Then
579:         For Each objWMIIInstance In objWMIIInstances
580:             strDNSServerName = objWMIIInstance.Name
581:         Next
582:     End If
...
586:     '
587:     If boolSRVView Then
588:         Set objWMIIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''' &_
589:                                         strDNSServerName & '''")
...
592:         WScript.Echo "- " & objWMIIInstance.Name & " " & String (60, "-")
593:         Set objWMIPropertySet = objWMIIInstance.Properties_
594:         For Each objWMIProperty In objWMIPropertySet
595:             DisplayFormattedProperty objWMIIInstance, _
596:                                         " " & objWMIProperty.Name, _
597:                                         objWMIProperty.Name, _
598:                                         Null
599:         Next
...
602:         WScript.Echo
603:
604:         WScript.Echo "- Hosted zones " & String (60, "-")
605:
606:         Set objWMIAssocInstances = objWMIServices.ExecQuery -
607:                                         ("Associators of {'" & _
608:                                         objWMIIInstance.Path_.RelPath & _
609:                                         "'} Where AssocClass=" & _
610:                                         "MicrosoftDNS_ServerDomainContainment")
611:         For Each objWMIAssocInstance In objWMIAssocInstances
612:             If objWMIAssocInstance.Path_.Class = "MicrosoftDNS_Zone" Then
613:                 WScript.Echo " " & objWMIAssocInstance.Name
614:             End If
615:         Next
...
619:     End If
620:     If boolSRVStart Then
621:         Set objWMIIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''' &_
622:                                         strDNSServerName & '''")
...
625:         objWMIIInstance.StartService()
...
628:         WScript.Echo "DNS Server started."
...
631:     End If
632:     If boolSRVStop Then
633:         Set objWMIIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''' &_
634:                                         strDNSServerName & '''")
...
637:         objWMIIInstance.StopService()
...
640:         WScript.Echo "DNS Server stopped."
...
643:     End If
644:     If boolSRVScavenging Then
645:         Set objWMIIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''' & _
```

```
646:                               strDNSServerName & "")  
...:  
648:     objWMIInstance.StartScavenging()  
...:  
650:     WScript.Echo "DNS Server Scavenging requested."  
...:  
652: End If  
653: If boolBootMethod Then  
654:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=' " &  
655:                                                 strDNSServerName & "'")  
...:  
658:     objWMIInstance.bootMethod = intBootMethod  
659:  
660:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
661:                             wbemFlagReturnWhenComplete)  
...:  
664:     WScript.Echo "DNS Server boot method updated."  
...:  
667: End If  
668: If boolEventLogLevel Then  
669:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=' " &  
670:                                                 strDNSServerName & "'")  
...:  
673:     objWMIInstance.EventLogLevel = intEventLogLevel  
674:  
675:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
676:                             wbemFlagReturnWhenComplete)  
...:  
679:     WScript.Echo "DNS Server event log level updated."  
...:  
682: End If  
683: If boolForwarders Then  
684:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=' " &  
685:                                                 strDNSServerName & "'")  
...:  
688:     objWMIInstance.Forwarders = arrayForwarders  
689:  
690:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
691:                             wbemFlagReturnWhenComplete)  
...:  
694:     WScript.Echo "DNS Server fowarders updated."  
...:  
697: End If  
698: If boolSlave Then  
699:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=' " &  
700:                                                 strDNSServerName & "'")  
...:  
703:     objWMIInstance.IsSlave = boolIsSlave  
704:  
705:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
706:                             wbemFlagReturnWhenComplete)  
...:  
709:     WScript.Echo "DNS Server slave updated."  
...:  
712: End If  
713: If boolUpdateRecursion Then  
714:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=' " &  
715:                                                 strDNSServerName & "'")  
...:  
718:     objWMIInstance.NoRecursion = boolRecursion
```

```
719:  
720:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
721:                             wbemFlagReturnWhenComplete)  
...:  
724:     WScript.Echo "DNS Server recursion updated."  
...:  
727: End If  
728: If boolUpdateRoundRobin Then  
729:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''" & _  
730:                                         strDNSServerName & "")  
...:  
733:     objWMIInstance.RoundRobin = boolRoundRobin  
734:  
735:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
736:                             wbemFlagReturnWhenComplete)  
...:  
739:     WScript.Echo "DNS Server round robin updated."  
...:  
742: End If  
743: If boolIPAddresses Then  
744:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Server.Name=''" & _  
745:                                         strDNSServerName & "")  
...:  
748:     objWMIInstance.ListenAddresses = arrayIPAddresses  
749:  
750:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _  
751:                             wbemFlagReturnWhenComplete)  
...:  
754:     WScript.Echo "DNS Server listen IP addresses updated."  
...:  
757: End If  
758:  
...:  
...:  
...:
```

To manage, view, and update the DNS server information, the script always retrieves the DNS server instance first. Sample 3.66 contains several subportions, which are dedicated to a particular management task that can be performed with the DNS server. The tasks are the following:

- **Viewing the DNS information** (lines 587 through 619): These lines retrieve the DNS server instance and show all properties by using the `DisplayFormattedProperty()` function previously developed (lines 593 through 599). Because it is interesting to see the domain zones hosted on the DNS server, the script uses the associations in place to retrieve the domain list available from this server (lines 606 through 615). The following command-line would give an output such as:

```
1: C:\>WMIDNS.Wsf Server View  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5: - net-dpen6400a.LissWare.Net -----  
6: AddressAnswerLimit: ..... 0
```

```
7: AllowUpdate: ..... 2
8: AutoCacheUpdate: ..... FALSE
9: AutoConfigFileZones: ..... 1
10: BindSecondaries: ..... TRUE
11: BootMethod: ..... 2
12: DefaultAgingState: ..... FALSE
13: DefaultNoRefreshInterval: ..... 168
14: DefaultRefreshInterval: ..... 168
15: DisableAutoReverseZones: ..... FALSE
16: DisjointNets: ..... FALSE
17: DsAvailable: ..... TRUE
18: DsPollingInterval: ..... 300
19: DsTombstoneInterval: ..... 604800
20: EDnsCacheTimeout: ..... 86400
21: EnableDirectoryPartitions: ..... TRUE
22: EnableDnsSec: ..... 1
23: EnableEDnsProbes: ..... TRUE
24: EventLogLevel: ..... 2
25: ForwardDelegations: ..... 0
26: Forwarders: ..... 10.10.10.1
27: ForwardingTimeout: ..... 5
28: InstallDate: ..... 01-01-2000
29: IsSlave: ..... FALSE
30: ListenAddresses: ..... 10.10.10.3
31: LocalNetPriority: ..... TRUE
32: LogfileMaxSize: ..... 500000000
33: LogLevel: ..... 0
34: LooseWildcarding: ..... FALSE
35: MaxCacheTTL: ..... 86400
36: MaxNegativeCacheTTL: ..... 900
37: *Name: ..... net-dpen6400a.LissWare.Net
38: NameCheckFlag: ..... 2
39: NoRecursion: ..... FALSE
40: RecursionRetry: ..... 3
41: RecursionTimeout: ..... 15
42: RoundRobin: ..... TRUE
43: RpcProtocol: ..... -1
44: ScavengingInterval: ..... 0
45: SecureResponses: ..... TRUE
46: SendPort: ..... 0
47: ServerAddresses: ..... 10.10.10.3
48: Started: ..... TRUE
49: StartMode: ..... Automatic
50: Status: ..... OK
51: StrictFileParsing: ..... FALSE
52: UpdateOptions: ..... 783
53: Version: ..... 235274501
54: WriteAuthorityNS: ..... FALSE
55: XfrConnectTimeout: ..... 30
56:
57: - Hosted zones -----
58: _msdcs.LissWare.Net
59: 0.in-addr.arpa
60: 10.in-addr.arpa
61: 127.in-addr.arpa
62: 255.in-addr.arpa
63: LissWare.Net
64: LissWare.Net
```

We see the *MicrosoftDNS_Server* class properties from line 6 through 55 and the *MicrosoftDNS_Zone* associated instances from line 58 through 64.

- **Starting the DNS server service** (lines 620 through 631): The *MicrosoftDNS_Server* class is derived from the *CIM_Service* class. We have seen that this standard class exposes methods to start and stop a service. With the class inheritance mechanism of the CIM repository, the *MicrosoftDNS_Server* class also exposes these methods. This portion of the code makes use of the *StartService* method to start the DNS service (line 625).
- **Stopping the DNS server service** (lines 632 through 643): This portion of the code makes use of the *StopService* method coming from the *CIM_Service* class to stop the DNS service (line 637).
- **Scavenging the DNS server** (lines 644 through 652): Scavenging is the mechanism to perform cleanup and removal of stale DNS resource records, which can accumulate in zone data over time. By invoking the *StartScavenging* method of the *MicrosoftDNS_Server* class, it is possible to launch this cleanup process (line 648).
- **Setting the DNS boot method** (lines 653 through 667): A DNS server can be booted from a data file, the Windows Registry (default), or Active Directory. The *BootMethod* property defines the boot method to use (line 658). The boot method values are shown in Table 3.64.
- **Setting the DNS log level** (lines 668 through 682): The event logging level can be changed by setting the *EventLogLevel* property (line 673). The logging level values are also shown in Table 3.64. Do not confuse the event logging level with the debug logging level, for which values are also shown in Table 3.64.
- **Setting the DNS forwarders** (lines 683 through 697): Because several forwarders can be defined for one single DNS server, the *Forwarders* property accepts an array of IP addresses that correspond to the forwarder list (line 688).
- **Setting the DNS server as a slave** (lines 698 through 712): If the server must act as a slave, the *IsSlave* property of the *MicrosoftDNS_Server* class must be set to True (line 703).
- **Setting the DNS server recursion** (lines 713 through 727): To disable the recursion for the naming resolution, the *NoRecursion* prop-

→ **Table 3.64** Some DNS Server Property Values

Server boot	Value
File	1
Registry	2
Active Directory & Registry	3
Server Events Log Level	Value
None	0
Errors only	1
Errors & Warnings	2
All events	7
Debug Log Level	Value
Query	1
Notify	16
Update	32
Nonquery transactions	254
Questions	256
Answers	512
Send	4096
Receive	8192
UDP	16384
TCP	32768
All packets	65535
NT Directory Service write transaction	65536
NT Directory Service update transaction	131072
Full Packets	16777216
Write Through	2147483648

erty of the *MicrosoftDNS_Server* class must be set to True (line 718). This property corresponds to one of the advanced server options. Note that it is impossible to set forwarders when the recursion is disabled.

- **Setting the DNS server round-robin** (lines 728 through 742): If the DNS server must use the round-robin algorithm, which implements a load-balancing mechanism to share and distribute network resource loads, the *RoundRobin* property of the *MicrosoftDNS_Server* class must be set to True (line 733). This property corresponds to one of the advanced server options.
- **Setting the DNS server listening IP addresses** (lines 743 through 757): To define the IP address interfaces on which the DNS server must listen for DNS requests, the *ListenAddresses* property accepts an array of IP addresses that correspond to the interface IP address list (line 748).

The next portion of the script manages the DNS zones. This portion is divided into two subportions: the first one displays all details related to one specific zone (lines 762 through 802); the second one displays an information summary about all zones available in a DNS server (lines 804 through 831).

Sample 3.67 Viewing the DNS zones (Part IIa)

```

813:         strZoneTransfer = DecodeZoneTransfer (objWMIInstance.SecureSecondaries)
814:         strZoneNotify = DecodeZoneNotify (objWMIInstance.Notify)
815:         strZoneUpdate = DecodeZoneUpdate (objWMIInstance.AllowUpdate)
816:
817:         WScript.Echo String (25 - Len (objWMIInstance.Name), " ") & _
818:                         objWMIInstance.Name & _
819:                         String (15 - Len (strZoneType), " ") & _
820:                         strZoneType & _
821:                         String (20 - Len (strZoneUpdate), " ") & _
822:                         strZoneUpdate & _
823:                         String (20 - Len (strZoneTransfer), " ") & _
824:                         strZoneTransfer & _
825:                         String (20 - Len (strZoneNotify), " ") & _
826:                         strZoneNotify & _
827:                         String (14 - Len (objWMIInstance.Reverse), " ") & _
828:                         objWMIInstance.Reverse & _
829:                         String (10 - Len (objWMIInstance.Paused), " ") & _
830:                         objWMIInstance.Paused
831:         Next
...:
834:     End If
835: End If
...:
...:
...:
```

Let's start with the second portion of the code, since it shows a zone information summary (lines 804 through 831). This part of the code is only executed when no specific zone name is given on the command line (line 761). For example, the following command line would give:

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

Zone	Name	Type Updates	Transfer	Notify list ...
_msdcs.LissWare.Net AD Integrated	Only secure updates	To any servers	Only NS servers	...
0.in-addr.arpa	Primary	Not allowed	To any servers	Only NS servers
10.in-addr.arpa	Primary	Not allowed	None	Only listed servers
127.in-addr.arpa	Primary	Not allowed	To any servers	Only NS servers
255.in-addr.arpa	Primary	Not allowed	To any servers	Only NS servers
LissWare.Net	Primary	Not allowed	None	Only listed servers
LissWare.Net AD Integrated	Only secure updates	To any servers	Only NS servers	...

In opposition to the technique used to retrieve the existing zone list at the DNS server level (see Sample 3.66, lines 606 through 615), the script logic requests the list of all instances made from the *MicrosoftDNS_Zone* class (line 804). Of course, both methods are valid! This simply shows another scripting technique to retrieve the same information.

If the zone name is given on the command line, the script will retrieve all properties of the corresponding *MicrosoftDNS_Zone* instance. The output would be as follows:

```

1: C:\>WMIDNS.Wsf Zone View Name=LissWare.Net
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: - LissWare.Net -----
6: Aging: ..... TRUE
7: AllowUpdate: ..... Only secure updates
8: AutoCreated: ..... FALSE
9: AvailForScavengeTime: ..... 3515052
10: *ContainerName: ..... LissWare.Net
11: *DnsServerName: ..... net-dpen6400a.LissWare.Net
12: DsIntegrated: ..... TRUE
13: ForwarderSlave: ..... FALSE
14: ForwarderTimeout: ..... 0
15: InstallDate: ..... 01-01-2000
16: LastSuccessfulSoaCheck: ..... 0
17: LastSuccessfulXfr: ..... 0
18: *Name: ..... LissWare.Net
19: NoRefreshInterval: ..... 168
20: Notify: ..... Only NS servers
21: Paused: ..... FALSE
22: RefreshInterval: ..... 168
23: Reverse: ..... FALSE
24: SecureSecondaries: ..... To any servers
25: Shutdown: ..... FALSE
26: UseWins: ..... FALSE
27: ZoneType: ..... AD Integrated

```

More than simply retrieving information available from the existing zones, the script is also capable of performing the usual zone management tasks, such as creating a zone, updating the zone properties, and deleting a zone. These tasks are implemented in Sample 3.68.

Sample 3.68 Managing the DNS zones (Part IIb)

```

...:
...:
...:
835: End If
836: If boolZONECreate Then
837:     Set objWMIClass = objWMIServices.Get ("MicrosoftDNS_Zone")
...:
840:     Select Case intZoneType
841:         Case 1
842:             objWMIClass.CreateZone strZoneName, _
843:                             0, _
844:                             True, , , strAdminEmailName
845:         Case 2
846:             objWMIClass.CreateZone strZoneName, _
847:                             0, _
848:                             False, _
849:                             strZoneDataFile, _
850:                             ' _
851:                             strAdminEmailName
852:         Case 3
853:             objWMIClass.CreateZone strZoneName, _
854:                             1, _

```

```
855:                               False, _
856:                               strZoneDataFile, _
857:                               arrayIpAddresses, _
858:                               strAdminEmailName
859:             Case 4
860:                 objWMIClass.CreateZone strZoneName, _
861:                                         2, _
862:                                         False, _
863:                                         strZoneDataFile, _
864:                                         arrayIpAddresses, _
865:                                         strAdminEmailName
866:         End Select
...
869:     WScript.Echo "Zone '" & strZoneName & "' created."
...
872: End If
873: If boolZONEChange Then
874:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Zone." & _
875:                                         "ContainerName=''" & strZoneName & _
876:                                         "",DnsServerName=''" & strDNSServerName & _
877:                                         "",Name=''" & strZoneName & "")"
...
880:     Select Case intZoneType
881:         Case 1
882:             objWMIInstance.ChangeZoneType 0, True, , , strAdminEmailName
883:         Case 2
884:             objWMIInstance.ChangeZoneType 0, _
885:                                         False, _
886:                                         strZoneDataFile, _
887:                                         ,
888:                                         strAdminEmailName
889:         Case 3
890:             objWMIInstance.ChangeZoneType 1, _
891:                                         False, _
892:                                         strZoneDataFile, _
893:                                         arrayIpAddresses, _
894:                                         strAdminEmailName
895:         Case 4
896:             objWMIInstance.ChangeZoneType 2, _
897:                                         False, _
898:                                         strZoneDataFile, _
899:                                         arrayIpAddresses, _
900:                                         strAdminEmailName
901:     End Select
...
904:     objWMIInstance.AllowUpdate = intZoneUpdate
905:
906:     objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or _
907:                           wbemFlagReturnWhenComplete)
...
910:     WScript.Echo "Zone '" & strZoneName & "' updated."
...
913: End If
914: If boolZONEDelete Then
915:     Set objWMIInstance = objWMIServices.Get ("MicrosoftDNS_Zone." & _
916:                                         "ContainerName=''" & strZoneName & _
917:                                         "",DnsServerName=''" & strDNSServerName & _
918:                                         "",Name=''" & strZoneName & "")"
...
921:     objWMIInstance.Delete_
```

```
...:  
924:     WScript.Echo "Zone '" & strZoneName & "' deleted."  
...:  
927: End If  
928: If boolZONERefresh Then  
929:     Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." &  
930:                                         "ContainerName=''" & strZoneName &  
931:                                         ",DnsServerName=''" & strDNSServerName &  
932:                                         ",Name=''" & strZoneName & "")  
...:  
935:     objWMIService.ForceRefresh()  
...:  
938:     WScript.Echo "Zone '" & strZoneName & "' refreshed."  
...:  
941: End If  
942: If boolZONEPause Then  
943:     Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." &  
944:                                         "ContainerName=''" & strZoneName &  
945:                                         ",DnsServerName=''" & strDNSServerName &  
946:                                         ",Name=''" & strZoneName & "")  
...:  
949:     objWMIService.PauseZone()  
...:  
952:     WScript.Echo "Zone '" & strZoneName & "' paused."  
...:  
955: End If  
956: If boolZONEReLoad Then  
957:     Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." &  
958:                                         "ContainerName=''" & strZoneName &  
959:                                         ",DnsServerName=''" & strDNSServerName &  
960:                                         ",Name=''" & strZoneName & "")  
...:  
963:     objWMIService.ReloadZone()  
...:  
966:     WScript.Echo "Zone '" & strZoneName & "' reloaded."  
...:  
969: End If  
970: If boolZONEReset2 Then  
971:     Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." &  
972:                                         "ContainerName=''" & strZoneName &  
973:                                         ",DnsServerName=''" & strDNSServerName &  
974:                                         ",Name=''" & strZoneName & "")  
...:  
977:     objWMIService.ResetSecondaries arraySecondaries, _  
978:                                         intSecureSecondaries, _  
979:                                         arrayNotifyServers, _  
980:                                         intZoneNotify  
...:  
983:     WScript.Echo "Secondaries of zone '" & strZoneName & "' reset."  
...:  
986: End If  
987: If boolZONEResume Then  
988:     Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." &  
989:                                         "ContainerName=''" & strZoneName &  
990:                                         ",DnsServerName=''" & strDNSServerName &  
991:                                         ",Name=''" & strZoneName & "")  
...:  
994:     objWMIService.ResumeZone()  
...:  
997:     WScript.Echo "Zone '" & strZoneName & "' resumed."
```

```

....:
1000:    End If
1001:    If boolZONEUpdate Then
1002:        Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." & _
1003:                                         "ContainerName=''' & strZoneName & _"
1004:                                         "' ,DnsServerName=''' & strDNSServerName & _"
1005:                                         "' ,Name=''' & strZoneName & '''")
....:
1008:    objWMIService.UpdateFromDS()
....:
1011:    WScript.Echo "Zone '' & strZoneName & "' updated from DS."
....:
1014: End If
1015: If boolZONEWrite Then
1016:    Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Zone." & _
1017:                                         "ContainerName=''' & strZoneName & _"
1018:                                         "' ,DnsServerName=''' & strDNSServerName & _"
1019:                                         "' ,Name=''' & strZoneName & '''")
....:
1022:    objWMIService.WriteBackZone()
....:
1025:    WScript.Echo "Zone '' & strZoneName & "' WriteBack completed."
....:
1028: End If
1029: If boolCACHEClear Then
1030:    Set objWMIService = objWMIServices.Get ("MicrosoftDNS_Cache." & _
1031:                                         "ContainerName='..Cache' & _"
1032:                                         "' ,DnsServerName=''' & strDNSServerName & _"
1033:                                         "' ,Name='..Cache'''")
....:
1036:    objWMIService.ClearCache()
....:
1039:    WScript.Echo "DNS cache cleared."
....:
1042: End If
1043: If boolROOTHWrite Then
1044:    Set objWMIService = objWMIServices.Get ("MicrosoftDNS_RootHints." & _
1045:                                         "ContainerName='..RootHints' & _"
1046:                                         "' ,DnsServerName=''' & strDNSServerName & _"
1047:                                         "' ,Name='..RootHints'''")
....:
1050:    objWMIService.WriteBackRootHintDatafile()
....:
1053:    WScript.Echo "Roothints WriteBack completed."
....:
1056: End If
1057:
....:
....:
....:
....:

```

Here again, this portion can be divided into several subportions. Each portion corresponds to a management task that can be performed with a specific DNS zone. The tasks are as follows:

- **Creating a new zone** (lines 836 through 872): To create a zone we use the *CreateZone* method exposed by the *MicrosoftDNS_Zone* class. This method is invoked from the class instance (line 837), since the

CreateZone method is defined as a static method (you can check the *static* qualifier present in the method qualifiers). It is interesting to note that based on the zone type to be created, not all method parameters are required. When an Active Directory integrated zone is created, only the zone name (line 842), the value defining the zone type (line 843), the Boolean value setting the zone as an Active Directory Integrated zone (True), and the zone administrator's email address must be given. When a standard primary zone is created, we have the same set of parameters, but the Active Directory Integrated Boolean value is set to False (line 848) and the zone data file name is added (line 849) in the parameter list. For a secondary zone, again, we have the same set of parameters as for a primary zone creation, but the zone type value is different (line 854) and the primary IP address from which to transfer the zone is added (line 857). Basically, for a stub zone, the parameters are the same as for a secondary zone creation. Only the zone type is different (line 861). The values for the zone type are shown in Table 3.65.

Table 3.65 Some Zone Property Values

Zone type	Value
Primary or AID Integrated	0
Secondary	1
Stub	2
Zone transfer	Value
To any servers	0
Only NS servers	1
Only listed servers	2
No zone transfer	3
Zone notify	Value
No notification	0
Only NS servers	1
Only listed servers	2
Zone update	Value
Not allowed	0
Allow updates	1
Allow only secure updates	2

- Updating the zone type and its related information (lines 873 through 913): To update an existing zone, the script uses the *ChangeZoneType* method exposed by the *MicrosoftDNS_Zone* class. Basically, the method follows the same rules as the *CreateZone* method in the sense that not all method parameters are required, based on the zone type update to perform (lines 880 through 901). However, the method must be invoked from an instance representing the zone to be updated (lines 874 through 877). For example, this method can be used to convert an Active Directory integrated zone into a standard primary zone. It is interesting to note that the *ChangeZoneType*

method does not modify the zone update mode for the DNS dynamic updates. This property is stored in a specific property called *AllowUpdate*, which is updated after the method invocation in the script code (lines 904 through 907).

- **Deleting a zone** (lines 914 through 927): To delete a zone from a DNS server, there is no specific method to use. The scripting technique simply retrieves the zone instance (lines 915 through 918) and invokes the *Delete*_ method of the **SWBemObject** representing the zone instance (line 921).
- **Forcing the refresh of a zone** (lines 928 through 941): To refresh a DNS zone, the instance representing the zone must be retrieved (lines 929 through 932); next, the *ForceRefresh* method of the *Microsoft-DNS_Zone* instance must be invoked (line 935).
- **Pausing a zone** (lines 942 through 955): In the same way as with the zone refresh, to pause a DNS zone, the *PauseZone* method must be invoked (line 949) from an instance representing the zone to be paused.
- **Reloading a zone** (lines 956 through 969): To reload a zone, the technique is exactly the same. The script retrieves an instance of the zone (lines 957 through 960) and invokes the *ReloadZone* method (line 963).
- **Reconfiguring the secondary IP addresses** (lines 970 through 986): To update the configuration related to the secondary IP addresses, a specific method must be used. This method is called *ResetSecondaries* and allows the modification of several secondary server parameters. The first parameter accepted by the method contains the list of IP addresses representing the secondary servers (line 977). Next, the second method parameter defines how the zone transfer must be performed (line 978). The third method parameter contains a list of IP addresses that represents the servers to be notified for the zone updates (line 979). The final method parameter contains a value defining how the notification must be made (line 980). The various values used by this method are summarized in Table 3.65.
- **Resuming a zone** (lines 987 through 1000): Resuming a zone is the same as pausing a zone; the scripting technique is exactly the same.
- **Forcing a zone update** (lines 1001 through 1014): In the same way, from a scripting technique point of view, forcing a zone update is the same as reloading a zone.

- **Saving the zone back to the persistent storage** (lines 1015 through 1028): By invoking the *WriteBackZone* method, it is possible to force a save of the DNS zone information in a persistent storage. The persistent storage can be a zone file or Active Directory, based on the zone type.
- **Clearing the cache** (lines 1029 through 1042): The DNS cache is represented with the *MicrosoftDNS_Cache* class. The container name for the DNS cache is “..cache” (line 1031). By invoking the *ClearCache* method of the *MicrosoftDNS_Cache* instance, it is possible to clear the DNS cache content.
- **Saving the Root Hints to the used storage** (lines 1043 through 1056): The Root Hints are represented with the *MicrosoftDNS_RootHints* class. The container name for the DNS Root Hints zone is called “..RootHints” (line 1045). By invoking the *WriteBackRootHintDatafile* method of the *MicrosoftDNS_Zone* class, it is possible to save the Root Hints to the Root Hints data file.

By using the specific classes representing DNS records, it is possible to manage the records contained in the DNS zones. This is the purpose of Sample 3.69. The first operation is to retrieve the records contained in a zone. This operation is executed from line 1059 through 1098.

Sample 3.69 Managing the DNS records (Part III)

```
....:
....:
....:
1057:
1058: '
1059: If boolRECView Then
1060:     Set objWMIInstances = objWMIServices.ExecQuery ("Select * From " & _
1061:                                         strWMIClassRecordType & _
1062:                                         " Where ContainerName=' " & _
1063:                                         strDomainName & "'")
....:
1064: WScript.Echo "All '" & DecodeRecordClassString (strWMIClassRecordType) & _
1065:             "' records of domain '" & _
1066:             strDomainName & _
1067:             "'." & vbCRLF
1068:
1069:
1070:
1071: For Each objWMIInstance In objWMIInstances
1072:     Set objWMIPropertySet = objWMIInstance.Properties_
1073:     For Each objWMIProperty In objWMIPropertySet
1074:         Select Case objWMIProperty.Name
1075:             Case "ContainerName"
1076:             Case "TextRepresentation"
1077:             Case "InstallDate"
1078:             Case "DomainName"
1079:             Case "RecordData"
```

```

1080:             Case "RecordClass"
1081:                 DisplayFormattedProperty objWMIInstance, _
1082:                     " " & objWMIProperty.Name, _
1083:                     DecodeRecordClass (objWMIProperty.Value), _
1084:                     Null
1085:             Case Else
1086:                 DisplayFormattedProperty objWMIInstance, _
1087:                     " " & objWMIProperty.Name, _
1088:                     objWMIProperty.Name, _
1089:                     Null
1090:             End Select
1091:         Next
1092:     WScript.Echo
....:
1095:     Next
....:
1098: End If
1099: If boolRECCreate Then
1100:     Set objWMIClass = objWMIServices.Get (strWMIClassRecordType)
....:
1103:     objWMIClass.CreateInstanceFromPropertyData strDNSServerName, _
1104:                     strDomainName, _
1105:                     strHostOwner & _
1106:                     "." & strDomainName, _
1107:                     intRecordClass, _
1108:                     intTTL, _
1109:                     strHostAddress
....:
1112: WScript.Echo """ & strRecordType & "' record for '" & _
1113:             strHostOwner & "." & strDomainName & "' created."
....:
1116: End If
1117: If boolRECDelete Then
1118:     Set objWMIInstance = objWMIServices.Get (strWMIClassRecordType & _
1119:                     ".ContainerName=''" & strDomainName & _
1120:                     "",DnsServerName=''" & strDNSServerName & _
1121:                     "",DomainName=''" & strDomainName & _
1122:                     "",OwnerName=''" & strHostOwner & "." & strDomainName & _
1123:                     "",RecordClass=" & intRecordClass & _
1124:                     ",RecordData=" & strHostAddress & "")")
....:
1127:     objWMIInstance.Delete_
....:
1130: WScript.Echo """ & strRecordType & "' record for '" & _
1131:             strHostOwner & "." & strDomainName & "' deleted."
....:
1134: End If
1135:
1136: ]]>
1137: </script>
1138: </job>
1139:</package>
```

To retrieve the records available in a zone, the script performs a WQL query with some criteria to ensure that only the records of the given zone are retrieved (lines 1060 through 1063). We see in the query that the WMI record class name is stored in a variable (line 1061). If the Rtype parameter given on the command-line is an asterisk (*), then the class name used is the

MicrosoftDNS_ResourceRecord class. Because the *MicrosoftDNS_ResourceRecord* class is a superclass for all record type classes, the use of this class in the WQL query will retrieve all records available in the zone. If the Rtype parameter given on the command-line corresponds to a specific record type (i.e., A record or CNAME record), then only this record type will be viewed, and the class used in the WQL query will be the one corresponding to the record type (see Table 3.63 to get a list of the record type WMI classes). This class selection, based on the record type, is made during the command-line parameters parsing. The following command line will list all A records available in the LissWare.Net domain:

```
1:  C:\>WMIDNS.Wsf Record View RType=A DomainName=LissWare.Net
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  All 'A' records of domain 'LissWare.Net'.
6:
7:  *DnsServerName: ..... net-dpen6400a.LissWare.Net
8:  IPAddress: ..... 10.10.10.3
9:  *OwnerName: ..... LissWare.Net
10: RecordClass: ..... IN
11: TTL: ..... 600
12:
13: *DnsServerName: ..... net-dpen6400a.LissWare.Net
14: IPAddress: ..... 10.10.10.253
15: *OwnerName: ..... Cisco-Brussels.LissWare.Net
16: RecordClass: ..... IN
17: TTL: ..... 3600
18:
19: *DnsServerName: ..... net-dpen6400a.LissWare.Net
20: IPAddress: ..... 10.10.10.3
21: *OwnerName: ..... DomainDnsZones.LissWare.Net
22: RecordClass: ..... IN
23: TTL: ..... 600
24:
25: *DnsServerName: ..... net-dpen6400a.LissWare.Net
26: IPAddress: ..... 10.10.10.3
27: *OwnerName: ..... ForestDnsZones.LissWare.Net
28: RecordClass: ..... IN
29: TTL: ..... 600
30:
31: *DnsServerName: ..... net-dpen6400a.LissWare.Net
32: IPAddress: ..... 192.10.10.3
33: *OwnerName: ..... ForestDnsZones.LissWare.Net
34: RecordClass: ..... IN
35: TTL: ..... 600
36:
37: *DnsServerName: ..... net-dpen6400a.LissWare.Net
38: IPAddress: ..... 10.10.10.3
39: *OwnerName: ..... net-dpen6400a.LissWare.Net
40: RecordClass: ..... IN
41: TTL: ..... 3600
42:
43: *DnsServerName: ..... net-dpen6400a.LissWare.Net
44: IPAddress: ..... 192.10.10.3
45: *OwnerName: ..... net-dpep6400.Emea.LissWare.Net
```

```
46:     RecordClass: ..... IN
47:     TTL: ..... 3600
48:
49:     *DnsServerName: ..... net-dpen6400a.LissWare.Net
50:     IPAddress: ..... 10.10.10.100
51:     *OwnerName: ..... XP-PRO01.LissWare.Net
52:     RecordClass: ..... IN
53:     TTL: ..... 1200
```

To create a new DNS record, the *CreateInstanceFromPropertyData* method is used in the script code (lines 1099 through 1116). This method is specific to the DNS record type and is not available at the level of the *MicrosoftDNS_ResourceRecord* class. Note that it is possible to use a text representation of the record to create a new record in the DNS zone. In such a case, the *CreateFromTextRepresentation* method of the *MicrosoftDNS_ResourceRecord* class can be used. Using one method or the other will always create a DNS record. However, based on the situation, one method will be easier than the other. By using the generic *CreateFromTextRepresentation* method of the *MicrosoftDNS_ResourceRecord* class, it is possible to create any record type with one single parameter, which is the text representation of the record type. This makes the scripting technique easier, since only one method with one single parameter must be used. However, the correct text representation must be passed to the method. As an example, we have the text representation of an A record and a CNAME record:

```
Net-dpen6400a.LissWare.Net IN A 10.10.10.3
www.LissWare.Net IN CNAME Net-dpen6400a.LissWare.Net.
```

The second method, called *CreateInstanceFromPropertyData*, requires specific parameters determined by the record type to be created. In such a case, the scripting technique becomes specific to the record type but does not require the DNS record string representation. Currently, the script uses the *CreateInstanceFromPropertyData* method defined at the level of the *MicrosoftDNS_Atype* class. Because the *MicrosoftDNS_CNAMEType* class exposes the same method name with the same number of parameters, it is possible to create transparently with the *CreateInstanceFromPropertyData* method A and CNAME records, provided that the correct parameters are passed to the method. Although the number of parameters is the same, the meaning of all parameters between an A record and a CNAME record is not exactly the same. The two following command-line samples show the parameter difference:

```
C:\>WMIDNS.Wsf Record Create RType=A DomainName=LissWare.Net Host=Net-dpen6400a Class=IN
TTL=3600 HostAddress=10.10.10.7

C:\>WMIDNS.Wsf Record Create RType=CNAME DomainName=LissWare.Net Host=www Class=IN
TTL=7200 HostAddress=Net-dpen6400a.LissWare.Net.
```

We see that most parameters are the same. However, we notice that the *HostAddress* is an IP address when creating an A record and that *HostAddress* is an FQDN when creating a CNAME record. The record class remains the same in both cases. The values used to define the record class during the *CreateInstanceFromPropertyData* method invocation are listed in Table 3.66.

Table 3.66*The DNS Record Class Values*

Record class	Value
IN (INTERNET)	1
CS (CSNET)	2
CH (CHAOS)	3
HS (HESIOD)	4

The scripting technique to delete a record is very similar to the technique used to delete a zone. The first step is to retrieve an instance of the record (lines 1118 through 1124) and then to invoke the *Delete_* method of the **SWBemObject** representing the record instance (line 1127). To delete a DNS record, the following command-line parameter can be used:

```
C:> \WMIDNS.Wsf Record Delete RType=A DomainName=LissWare.Net Host=Net-dpen6400a Class=IN
      TTL=3600 HostAddress=10.10.10.7
```

3.7.5 SNMP providers

The WMI *SNMP* providers are designed to integrate the Simple Network Management Protocol (SNMP) management information in WMI, which means that it is possible to seamlessly manage SNMP devices from WMI. By default, none of the *SNMP* providers is installed on any Windows platform! So, to install these providers, you should perform the following tasks:

- **For Windows XP and Windows Server 2003:** From the control panel, select Add/Remove Programs. Next, select Add/Remove Windows Components, and then in the Windows Components Wizard, select Management and Monitoring Tools (see Figure 3.39).

Finally, select WMI *SNMP* providers, then click OK. Follow the steps in the wizard to complete the installation.

- **For Windows 2000:** Run the *SNMP* provider Setup program, **Wbemsnmp.exe**, from the **System32\wbem** directory or the **\i386** directory of the Windows 2000 installation CD.
- **For Windows NT 4.0:** Install the *SNMP* provider when installing the WMI core component or from the Internet at <http://www.microsoft.com/downloads/details.aspx?FamilyID=f8130806-2589-46b3-b472-26b816776f31&DisplayLang=en>.

Figure 3.39
Adding the SNMP providers under Windows Server 2003.



Note that under Windows 95, 98, and Millennium, the WMI *SNMP* provider cannot be installed or run on this operating system.

The *SNMP* providers are made up of one instance provider, one class provider, and two event providers. (See Table 3.67.)

Table 3.67 The *SNMP Providers 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
SNMP Providers																
MS_SNMP_CLASS_PROVIDER	Root/snmp/localhost	X						X	X			X	X	X	X	X
MS_SNMP_ENCAPSULATED_EVENT_PROVIDER	Root/snmp/localhost					X						X	X	X	X	X
MS_SNMP_INSTANCE_PROVIDER	Root/snmp/localhost		X					X	X	X	X	X	X	X	X	X
MS_SNMP_REFERENT_EVENT_PROVIDER	Root/snmp/localhost					X						X	X	X	X	X

Once the *SNMP* providers are installed, by default two new namespaces are created: the **Root\SNMP\SMIR** and the **Root\SNMP\LocalHost**. The **Root\SNMP\SMIR** namespace contains the *SNMP* schema database, which is called the *SNMP Module Information Repository* (*SMIR*). *SNMP* definitions, called the *Structure of Management Information* (*SMI*), are available from *Management Information Base* (*MIB*) files and represent the

real-world manageable entities as defined by SNMP. MIB files use a standard notation (Abstract Syntax Notation 1 or ASN.1) and a number of macro definitions that serve as templates to describe real-world manageable entities. We will not go into the syntax details of MIB files, but it is important to realize that MIB files are the primary source of information used by WMI to represent these real-world SNMP manageable entities in the CIM repository. By default, the WMI installation only loads MIB definitions from the RFC 1213 and RFC 1215 MIB files that define standard MIB to manage IP networks. (See Table 3.68.)

Table 3.68 The SNMP Classes Available in the Root\SNMP\SMIR Namespaces

Name	Type	Comments
SNMP_RFC1213_MIB_atTable	Dynamic	The Address Translation tables contain the NetworkAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty (i.e., has zero entries).
SNMP_RFC1213_MIB_egp	Dynamic	The Exterior Gateway Protocol (EGP) group represents information about: The number of EGP messages received without error, the number of EGP messages received that proved to be in error, the total number of locally generated EGP messages, and the number of locally generated EGP messages not sent due to resource limitations within an EGP entity.
SNMP_RFC1213_MIB_egpNeighTable	Dynamic	The Exterior Gateway Protocol (EGP) neighbor table contains information about this entity's EGP neighbors.
SNMP_RFC1213_MIB_icmp	Dynamic	The ICMP group represents information about ICMP messages (number of messages, errors, unreachable destinations, exceeded time, redirections, etc.).
SNMP_RFC1213_MIB_ifTable	Dynamic	A list of interface entries. The number of entries is given by the value of ifNumber.
SNMP_RFC1213_MIB_interfaces	Dynamic	Represent the number of network interfaces (regardless of their current state) present on this system (ifNumber).
SNMP_RFC1213_MIB_ip	Dynamic	The IP group represents information about IP Forwarding, IP TTL, IP traffic, IP errors, Datagrams, packets discarded, etc.
SNMP_RFC1213_MIB_ipAddrTable	Dynamic	The table of addressing information relevant to this entity's IP addresses.
SNMP_RFC1213_MIB_ipNetToMediaTable	Dynamic	The IP Address Translation table used for mapping from IP addresses to physical addresses.
SNMP_RFC1213_MIB_ipRouteTable	Dynamic	The entity's IP Routing table.
SNMP_RFC1213_MIB_snmp	Dynamic	The entity's SNMP information.
SNMP_RFC1213_MIB_system	Dynamic	The entity's system information (contact, location, etc.).
SNMP_RFC1213_MIB_tcp	Dynamic	The TCP group represents information about TCP stack (open connections, errors, packet transmit and receive, etc.).
SNMP_RFC1213_MIB_tcpConnTable	Dynamic	Table containing active TCP connections information.
SNMP_RFC1213_MIB_udp	Dynamic	The UDP group represent information about UDP stack (datagrams, errors, packet transmit and receive, etc.)
SNMP_RFC1213_MIB_udpTable	Dynamic	Table containing UDP listener information.
Snmpv1ExtendedNotification	Extrinsic (Enterprise non-specific)	SNMP version 1 Enterprise nonspecific events.
Snmpv1Notification	Extrinsic (Enterprise non-specific)	SNMP version 1 Enterprise nonspecific events.
Snmpv2ExtendedNotification	Extrinsic (Enterprise non-specific)	SNMP version 2 Enterprise nonspecific events.
Snmpv2Notification	Extrinsic (Enterprise non-specific)	SNMP version 2 Enterprise nonspecific events.
SnmpAuthenticationFailureExtendedNotification	Extrinsic (Generic)	Notification representing an authentication failure.
SnmpAuthenticationFailureNotification	Extrinsic (Generic)	Notification representing an authentication failure.
SnmpColdStartExtendedNotification	Extrinsic (Generic)	Notification representing an SNMP device cold start.
SnmpColdStartNotification	Extrinsic (Generic)	Notification representing an SNMP device cold start.
SnmpEGPNeighborLossExtendedNotification	Extrinsic (Generic)	Notification representing an Exterior Gateway Protocol (EGP) neighbor loss.
SnmpEGPNeighborLossNotification	Extrinsic (Generic)	Notification representing an Exterior Gateway Protocol (EGP) neighbor loss.
SnmpExtendedNotification	Extrinsic (Generic)	The SnmpExtendedNotification class is the base class for any class mapped from the NOTIFICATION-TYPE macro to a CIM class by the SNMP Provider.
SnmpLinkDownExtendedNotification	Extrinsic (Generic)	Notification representing an interface link down.
SnmpLinkDownNotification	Extrinsic (Generic)	Notification representing an interface link down.
SnmpLinkUpExtendedNotification	Extrinsic (Generic)	Notification representing an interface link up.
SnmpLinkUpNotification	Extrinsic (Generic)	Notification representing an interface link up.
SnmpNotification	Extrinsic (Generic)	The SnmpNotification class is the base class for any class mapped from the NOTIFICATION-TYPE macro to an encapsulated CIM class by the SNMP Provider.
SnmpWarmStartExtendedNotification	Extrinsic (Generic)	Notification representing an SNMP device warm start.
SnmpWarmStartNotification	Extrinsic (Generic)	Notification representing an SNMP device warm start.

Both RFC 1213 and RFC 1215 MIB are loaded in the SNMP Module Information Repository (`Root\SNMP\SMIR`) namespace and are represented in the form of WMI classes, as shown in Figure 3.40.



Figure 3.40 The SNMP Module Information Repository classes in `Root\SNMP\SMIR`.

Of course, since these classes represent SNMP information, we leave the world of WMI to go into the world of SNMP. Although some class names reflect very well the information they provide, you can refer to RFC 1213 and RFC 1215 to get more information (see <http://www.faqs.org/rfcs/rfc1213.html> and <http://www.faqs.org/rfcs/rfc1215.html>). We have dynamic classes, which are used to represent SNMP instances from RFC 1213 (left side of Figure 3.40), and extrinsic event classes, which represent SNMP event classes from RFC 1215, usually called SNMP traps or notifications (right side of Figure 3.40).

The *SNMP* dynamic class provider uses the SNMP Module Information Repository to create and retrieve class definitions. It is important to note that the *SMIR* namespace does not contain the *SNMP* provider registration instances (made from the `_Win32Provider` class). This namespace is just

used as a repository containing the SNMP MIB definitions to be used by WMI. Of course, there are a lot of things to say about the WMI *SNMP* providers and their integration with WMI. However, in the context of this book, instead of doing long theoretical descriptions, we will show you how you can take advantage of this integration. And by practicing, you will understand how things work together.

3.7.5.1 Accessing SNMP data

To access SNMP information available from a Windows system, ensure that the SNMP service is properly configured and running, since it implements the SNMP agent contacted to provide information (see Figure 3.41).

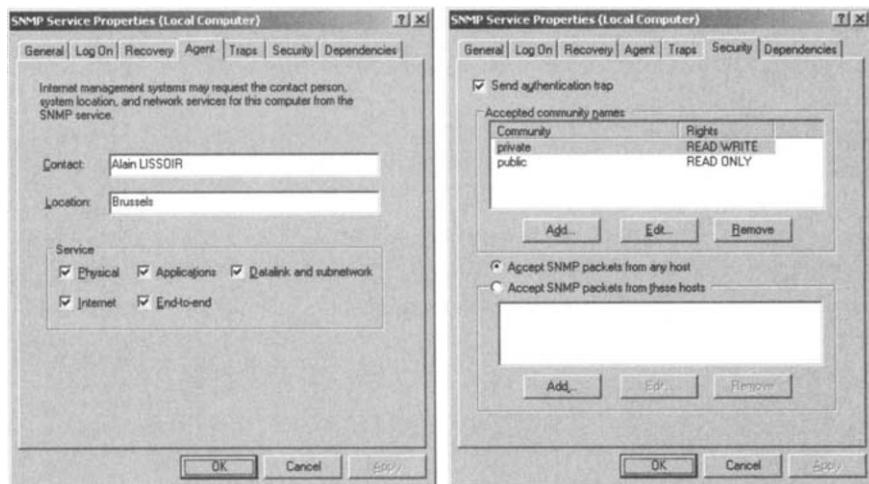
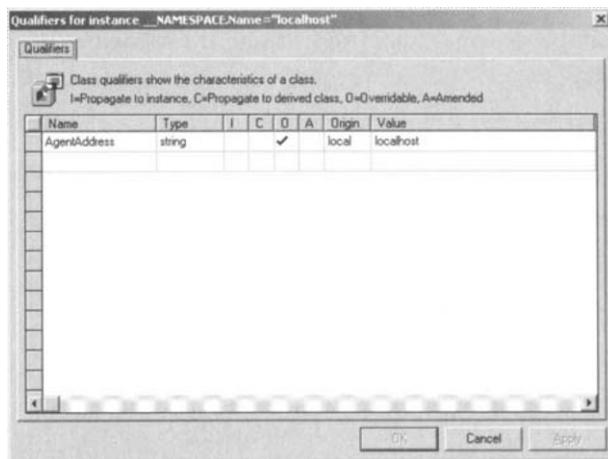


Figure 3.41 The SNMP service configuration.

Next, we can use the `Root\SNMP\localhost` namespace created during the installation of the WMI *SNMP* providers. This namespace is an SNMP proxy namespace and contains registration instances of the WMI *SNMP* providers (made from the `_Win32Provider` class). By default, all WMI *SNMP* providers are available from this namespace (instance, class, and event providers).

Each SNMP proxy WMI namespace has a set of qualifiers that describe the transport characteristics for communicating with an SNMP agent. In the case of the local system, there is one qualifier, called `AgentAddress`, defining the host name of the system, which in this case is the localhost (see Figure 3.42).

Figure 3.42
The
Root\SNMP\localhost namespace qualifier.



The *AgentAddress* is the only mandatory qualifier; however, there are other qualifiers defining other SNMP communication parameters. Table 3.69 summarizes the available qualifiers.

Table 3.69 *The Proxy Namespace Qualifiers to Define SNMP Transport Characteristics*

AgentAddress	CIM_STRING	Mandatory	Transport address associated with the SNMP agent, such as an IP address or DNS name. AgentAddress should be set to a valid unicast host address or a domain host name that can be resolved through the operating system's domain-name resolution process. There is no default value.
AgentTransport	CIM_STRING	Optional	Transport protocol used to communicate with the SNMP agent. Currently the only valid values are Internet Protocol (IP) and Internet Packet Exchange (IPX). The default value is IP.
AgentReadCommunityName	CIM_STRING	Optional	Variable-length octet string that is used by the SNMP agent to authenticate an SNMP Protocol Data Unit (PDU) during a read operation (SNMP GET/GET NEXT requests). The community name must be mapped to a Unicode string and is limited to alphanumeric characters. The default value is public.
AgentWriteCommunityName	CIM_STRING	Optional	Variable-length octet string that is used by the SNMP agent to authenticate an SNMP Protocol Data Unit (PDU) during a read operation (SNMP SET request). The community name must be mapped to a Unicode string and is limited to alphanumeric characters. The default value is public.
AgentRetryCount	CIM_SINT32	Optional	Number of times a single SNMP request can be retried when there is no response from the SNMP agent before the request is deemed to have failed. AgentRetryCount must be set to an integer between 0 and 2^32-1. The default value is 1.
AgentRetryTimeout	CIM_SINT32	Optional	Time in milliseconds before the SNMP Protocol Data Unit (PDU) is considered to have been dropped. This is the number of milliseconds to wait for a response to an SNMP request sent to the SNMP agent. AgentRetryTimeout must be set to an integer between 0 and 2^32-1. The default value is 500.
AgentVarBindsPerPdu	CIM_SINT32	Optional	Maximum number of variable bindings contained within a single SNMP PDU. Every SNMP request sent to the SNMP agent contains one or more SNMP variables. This parameter specifies the largest number of variables that can be included in a single request. AgentVarBindsPerPdu can be set to 0 to cause the implementation to determine the optimal variable bindings or to an integer between 1 and 2^32-1. Note that while increasing this value can improve performance, some agents and networks cannot deal with the resulting request. The default value is 10.
AgentFlowControlWindowSize	CIM_SINT32	Optional	Maximum number of concurrently outstanding SNMP requests that can be sent to this agent while a response has not yet been received. An SNMP request is considered outstanding until a PDU response has been received or it has timed out. A large window size generally improves performance, but some agents might not work well under a heavy load. AgentFlowControlWindowSize can be set to 0 to indicate an infinite window size or to an integer between 1 and 2^32-1. The default value is 10.
AgentSNMPVersion	CIM_SINT32	Optional	Version of the SNMP protocol to be used when communicating with the SNMP agent. Currently, the only valid values are 1 (for SNMPv1) and 2C (for SNMPv2C). The default value is 1.
Correlated	CIM_BOOLEAN	Optional	Set to True, defines the set of classes that a given SNMP agent is known to support at the time the enumeration occurs. Set to False, enumeration returns all classes present within the SMIr namespace (noncorrelated), regardless of whether the agent device supports them or not.

As you can see, you can use the *AgentReadCommunityName* to define the SNMP community string (public by default) or the *AgentTransport* to

define the protocol type (IP by default, but it can be IPX). Let's take an example where a computer is using the following IP address and mask settings:

```
C:\>ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

  Connection-specific DNS Suffix . :
  IP Address . . . . . : 10.10.10.3
  Subnet Mask . . . . . : 255.0.0.0
  Default Gateway . . . . . : 10.10.10.254
```

By using Sample 3.70, we can obtain from WMI and via SNMP the same type of information.

→ **Sample 3.70** *Obtaining localhost IP addresses from WMI via SNMP*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
10:
11:    <?job error="True" debug="True" ?>
12:
13:    <script language="VBScript" src=..\Functions\DisplayFormattedPropertiesFunction.vbs" />
14:    <script language="VBScript" src=..\Functions\DisplayFormattedPropertyFunction.vbs" />
15:
16:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
17:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
18:
19:    <script language="VBScript">
20:      <![CDATA[
.:
24:      Const cComputerName = "LocalHost"
25:      Const cWMINameSpace = "Root/SNMP/localhost"
26:      Const cWMIClass = "SNMP_RFC1213_MIB_ipAddrTable"
.:
33:      objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
34:      objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
35:      Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
36:
37:      Set objWMIInstances = objWMIServices.InstancesOf(cWMIClass)
38:
39:      For Each objWMIInstance In objWMIInstances
40:        DisplayFormattedProperties objWMIInstance, 0
41:        WScript.Echo
42:      Next
.:
47:    ]]>
48:  </script>
49: </job>
50:</package>
```

Sample 3.70 contains nothing complicated from a scripting point of view. We developed this script throughout the first book, *Understanding WMI Scripting*, when discovering the WMI Scripting API (see Sample 4.6 in the appendix). This demonstrates how transparent SNMP access is from WMI. We just need to access the appropriate WMI namespace (line 25) and request the *SNMP RFC1213 MIB_ipAddrTable* class, which represents the requested SNMP information (line 26). The output would be as follows:

```
1: C:\>GetSingleInstanceWithAPI.wsf
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
6: - SNMP RFC1213 MIB_ipAddrTable -----
7: *ipAdEntAddr: ..... 10.10.10.3
8: ipAdEntBcastAddr: ..... 1
9: ipAdEntIfIndex: ..... 33554434
10: ipAdEntNetMask: ..... 255.0.0.0
11: ipAdEntReasmMaxSize: ..... 65535
12:
13:
14: - SNMP RFC1213 MIB_ipAddrTable -----
15: *ipAdEntAddr: ..... 127.0.0.1
16: ipAdEntBcastAddr: ..... 1
17: ipAdEntIfIndex: ..... 1
18: ipAdEntNetMask: ..... 255.0.0.0
19: ipAdEntReasmMaxSize: ..... 65535
```

Accessing standard SNMP information from the local system is pretty straightforward, but how do you access SNMP information from a system other than the local host? We have two solutions: We can modify Sample 3.70 and make use of an **SWBemNamedValueSet** object, or we can create a new proxy namespace for that particular device or host. For this exercise, we will access via SNMP, a Cisco router 2503 loaded with the Cisco Internetwork Operating System (IOS) Software version 11.2(3)P. If you plan to use another SNMP-enabled device, please refer to the device specifications to determine if it supports RFC 1213 and RFC 1215. For the Cisco IOS, these RFCs are supported from version 10.2 or later.

First, the Cisco router must be SNMP enabled. If you are not familiar with Cisco devices, please contact your network administrator to get some help for the Cisco configuration. Figure 3.43 shows the basic commands that are required to SNMP enable a Cisco router 2500. The router can be easily configured via a TELNET session, assuming it is preconfigured for TELNET access and you have the required credentials.

```

User Access Verification
Password:
Brussels>en
Password:
Brussels#conf t
Enter configuration commands, one per line. End with CNTL/Z.
Brussels(config)#snmp-server community public RO
Brussels(config)#snmp-server community private RW
Brussels(config)#snmp-server trap-authentication
Brussels(config)#snmp-server location Brussels
Brussels(config)#snmp-server contact Alain LISSOIR - HPCI - Technology Leadership Group
Brussels(config)#snmp-server enable traps
Brussels(config)#snmp-server host 10.10.10.3 public
Brussels(config)#
Brussels#exit
Brussels#wr
Building configuration...
[OK]
Brussels#-

```

Figure 3.43 Enabling SNMP on your Cisco router.

As you can see in Figure 3.43, you can easily recognize the SNMP community string for the read-only operations (RO=public) and read/write operations (RW=private). Make sure that you use the correct community string for the SNMP agent; otherwise, you won't be able to access the Cisco router. Note that the read/write community string is optional in this example (actually, we will also send SNMP commands to the device in section 3.7.5.5, "Sending SNMP commands"). Again, request the help of your network administrator to correctly configure the SNMP information in your Cisco router, and, if you configure this in production, make sure that you respect the security policies of your company.

To access a remote SNMP device, we mentioned two options:

- Using an **SWBemNamedValueSet** object
- Creating a new WMI namespace

Let's start with the first one by using an **SWBemNamedValueSet** object. In this case, we do not create a dedicated namespace for the device as for the localhost. However, we reuse the existing namespace created for the localhost. This logic is shown in Sample 3.71.

Sample 3.71 Obtaining remote device IP addresses from WMI via SNMP

```

1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>

```

```
...  
13: <script language="VBScript" src=..\Functions\DisplayFormattedPropertiesFunction.vbs" />  
14: <script language="VBScript" src=..\Functions\DisplayFormattedPropertyFunction.vbs" />  
15:  
16: <object progid="WbemScripting.SWBemLocator" id="objWMILocator" reference="true"/>  
17: <object progid="WbemScripting.SWBemNamedValueSet" id="objWMINamedValueSet" />  
18: <object progid="WbemScripting.SWBemDateTime" id="objWMIDateTime" />  
19:  
20: <script language="VBScript">  
21: <![CDATA[  
...  
25: Const cComputerName = "LocalHost"  
26: Const cWMINameSpace = "Root/SNMP/localhost"  
27: Const cWMIClass = "SNMP_RFC1213_MIB_ipAddrTable"  
...  
34: objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault  
35: objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate  
36: Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")  
37:  
38: objWMINamedValueSet.Add "Correlate", True  
39: objWMINamedValueSet.Add "AgentAddress", "Cisco-Brussels.LissWare.Net"  
40: objWMINamedValueSet.Add "AgentFlowControlWindowSize", 3  
41: objWMINamedValueSet.Add "AgentReadCommunityName", "public"  
42: objWMINamedValueSet.Add "AgentRetryCount", 1  
43: objWMINamedValueSet.Add "AgentRetryTimeout", 500  
44: objWMINamedValueSet.Add "AgentVarBindsPerPdu", 10  
45: objWMINamedValueSet.Add "AgentWriteCommunityName", "private"  
46:  
47: Set objWMIInstances = objWMIServices.InstancesOf(cWMIClass, _  
48:                                         '  
49:                                         objWMINamedValueSet)  
50:  
51: For Each objWMIInstance In objWMIInstances  
52:     DisplayFormattedProperties objWMIInstance, 0  
53:     WScript.Echo  
54: Next  
...  
59: ]]>  
60: </script>  
61: </job>  
62:</package>
```

The script code is almost the same as Sample 3.70; only the initialization of the **SWBemNamedValueSet** is new (lines 38 through 45). The **SWBemNamedValueSet** object instantiated at line 17 is initialized with the qualifiers listed in Table 3.69. Note that the script uses all supported qualifiers. This is, of course, not mandatory, but it shows the complete coding technique. Passing the **SWBemNamedValueSet** object when requesting the collection of instances (line 49) supersedes the qualifiers defined at the **Root\SNMP\localhost** namespace level (see Figure 3.42).

We can see that the *AgentAddress* qualifier is set with the Fully Qualified Domain Name (FQDN) of the Cisco router (line 39), but it can also be the IP address. Depending on the Cisco router configuration, the output will look as follows:

```
1: C:\>GetSingleInstanceWithAPI2.wsf
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
6: - SNMP_RFC1213_MIB_ipAddrTable -----
7: *ipAdEntAddr: ..... 10.10.10.253
8: ipAdEntBcastAddr: ..... 1
9: ipAdEntIfIndex: ..... 1
10: ipAdEntNetMask: ..... 255.0.0.0
11: ipAdEntReasmMaxSize: ..... 18024
12:
13:
14: - SNMP_RFC1213_MIB_ipAddrTable -----
15: *ipAdEntAddr: ..... 192.10.10.253
16: ipAdEntBcastAddr: ..... 1
17: ipAdEntIfIndex: ..... 1
18: ipAdEntNetMask: ..... 255.255.255.0
19: ipAdEntReasmMaxSize: ..... 18024
20:
21:
22: - SNMP_RFC1213_MIB_ipAddrTable -----
23: *ipAdEntAddr: ..... 193.10.10.253
24: ipAdEntBcastAddr: ..... 1
25: ipAdEntIfIndex: ..... 1
26: ipAdEntNetMask: ..... 255.255.255.0
27: ipAdEntReasmMaxSize: ..... 18024
28:
29:
30: - SNMP_RFC1213_MIB_ipAddrTable -----
31: *ipAdEntAddr: ..... 194.10.10.253
32: ipAdEntBcastAddr: ..... 1
33: ipAdEntIfIndex: ..... 1
34: ipAdEntNetMask: ..... 255.255.255.0
35: ipAdEntReasmMaxSize: ..... 18024
36:
37:
38: - SNMP_RFC1213_MIB_ipAddrTable -----
39: *ipAdEntAddr: ..... 195.10.10.253
40: ipAdEntBcastAddr: ..... 1
41: ipAdEntIfIndex: ..... 2
42: ipAdEntNetMask: ..... 255.255.255.0
43: ipAdEntReasmMaxSize: ..... 18024
44:
45:
46: - SNMP_RFC1213_MIB_ipAddrTable -----
47: *ipAdEntAddr: ..... 200.10.10.1
48: ipAdEntBcastAddr: ..... 1
49: ipAdEntIfIndex: ..... 4
50: ipAdEntNetMask: ..... 255.255.255.0
51: ipAdEntReasmMaxSize: ..... 18024
```

The second solution is to create a dedicated namespace to access the Cisco router. This namespace can be created with a MOF file. In the %SystemRoot%\System32\WBEM directory, there is a MOF file called SNM-PREG.MOF. By using this MOF file as a template, it is very easy to obtain another MOF file to create the dedicated namespace to access the Cisco

router. We will call this new namespace the **Root\SNMP\CiscoBrussels** namespace. The MOF file creating this namespace with the required qualifiers is shown in Sample 3.72. From line 10 through 32, the MOF file creates the **Root\SNMP\CiscoBrussels** namespace with the SNMP qualifiers defining the communication parameters (lines 13 through 26). Next, the subsequent lines are the exact copy of the information coming from the **SNMPREG.MOF** and define the registration information of the **SNMP** providers for the dedicated Cisco router namespace (lines 36 through 101).

→ **Sample 3.72** A MOF file to create a dedicated namespace for an SNMP device

```
1:#pragma autorecover
2:
3:#pragma namespace ("\\\\.\\\\Root")
4:
5:instance of __Namespace
6:{ 
7:    Name = "SNMP" ;
8:} ;
9:
10:#pragma namespace ("\\\\.\\\\Root\\\\snmp")
11:
12:[
13:// IP address or DNS name.
14:AgentAddress ("Cisco-Brussels.LissWare.Net"),
15:// SNMP community name for SNMP GET/GETNEXT requests.
16:AgentReadCommunityName ("public"),
17:// SNMP community name for SNMP SET requests.
18:AgentWriteCommunityName ("private"),
19:// Number of SNMP PDU retries before transmission
20:AgentRetryCount (1),
21:// detects a 'No Response' from device.
22:AgentRetryTimeout (500),
23:// Maximum number of variable bindings contained within a single SNMP PDU.
24:AgentVarBindsPerPdu (10),
25:// Maximum number of outstanding SNMP PDUs that can be transmitted to an SNMP agent.
26:AgentFlowControlWindowSize (3)
27:]
28:
29:instance of __Namespace
30:{ 
31:    Name = "CiscoBrussels" ;
32:} ;
33:
34:#pragma namespace ("\\\\.\\\\Root\\\\snmp\\\\CiscoBrussels")
35:
36:instance of __Win32Provider as $PClass
37:{ 
38:    Name = "MS_SNMP_CLASS_PROVIDER";
39:    Clsid = "{70426720-F78F-11cf-9151-00AA00A4086C}";
40:    HostingModel = "NetworkServiceHost";
41:};
42:
43:instance of __ClassProviderRegistration
44:{
```

```
45:     Provider = $PClass;
46:     SupportsGet = TRUE;
47:     SupportsPut = FALSE;
48:     SupportsDelete = FALSE;
49:     SupportsEnumeration = TRUE;
50:
51:     QuerySupportLevels = NULL ;
52:
53:     ResultSetQueries = { "Select * From meta_class Where __this isa \"SnmpMacro\",
54:                           "Select * From meta_class Where __this isa \"SnmpNotification\",
55:                           "Select * From meta_class Where __this isa \"SnmpExtendedNotification\""
56:                         } ;
57: } ;
58:
59:instance of __Win32Provider as $PInst
60:{ 
61:     Name = "MS_SNMP_INSTANCE_PROVIDER";
62:     ClsId = "{1F517A23-B29C-11cf-8C8D-00AA00A4086C}";
63:     HostingModel = "NetworkServiceHost";
64:};
65:
66:instance of __InstanceProviderRegistration
67:{ 
68:     Provider = $PInst;
69:     SupportsGet = TRUE;
70:     SupportsPut = TRUE;
71:     SupportsDelete = TRUE;
72:     SupportsEnumeration = TRUE;
73:
74:     QuerySupportLevels = { "WQL:UnarySelect" } ;
75:};
76:
77:instance of __Win32Provider as $EventProv
78:{ 
79:     Name = "MS_SNMP_REFERENT_EVENT_PROVIDER";
80:     ClsId = "{9D5BED16-0765-11d1-AB2C-00C04FD9159E}";
81:     HostingModel = "LocalSystemHost";
82:};
83:
84:instance of __EventProviderRegistration
85:{ 
86:     Provider = $EventProv;
87:     EventQueryList = {"select * from SnmpExtendedNotification"} ;
88:};
89:
90:instance of __Win32Provider as $EncapEventProv
91:{ 
92:     Name = "MS_SNMP_ENCAPSULATED_EVENT_PROVIDER";
93:     ClsId = "{19C813AC-FEE7-11D0-AB22-00C04FD9159E}";
94:     HostingModel = "LocalSystemHost";
95:};
96:
97:instance of __EventProviderRegistration
98:{ 
99:     Provider = $EncapEventProv;
100:    EventQueryList = {"select * from SnmpNotification"};
101:};
```

We compile this MOF file with **MOFCOMP.EXE**, as follows:

```
C:\>MOFCOMP.Exe Cisco-Brussels-snmpreg.mof
Microsoft (R) 32-bit MOF Compiler Version 1.50.1085.0007
Copyright (c) Microsoft Corp. 1997-1999. All rights reserved.

Parsing MOF file: Cisco-Brussels-snmpreg.mof
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

Next, we can simply edit Sample 3.70 (“Obtaining localhost IP addresses from WMI via SNMP”) and change line 25:

```
25: Const cWMINameSpace = "Root/SNMP/LocalHost"
```

to the following line:

```
25: Const cWMINameSpace = "Root/SNMP/CiscoBrussels"
```

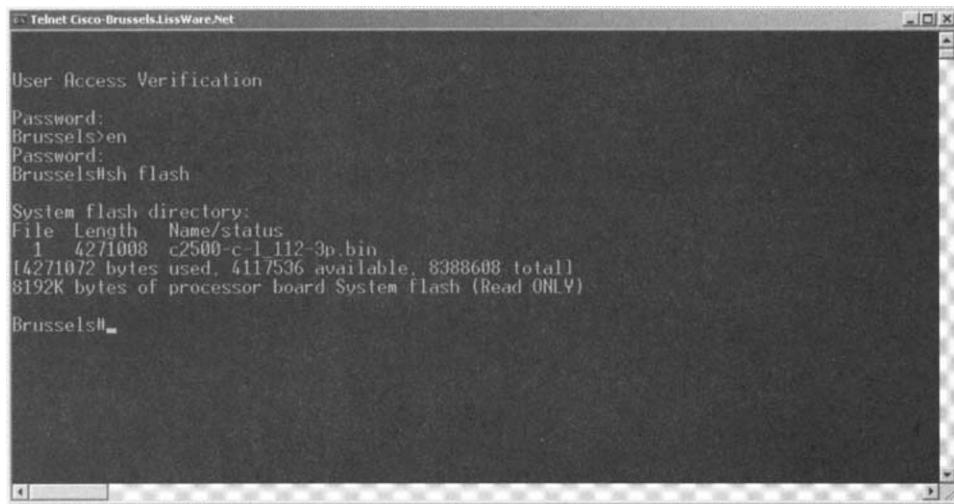
and we will obtain the exact same output as Sample 3.71 (“Obtaining remote device IP addresses from WMI via SNMP”), which was using an **SWBemNamedValueSet** object to pass the SNMP parameters.

3.7.5.2 **Accessing SNMP private MIB information**

Because we access a Cisco router, it is clear that we may find some specific SNMP information that is not defined in the standard RFC 1213. This SNMP information is called the private MIB information and is defined in other MIB files published by Cisco. You can get the Cisco MIB files from <http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml>. To access the private MIB information from WMI, it is necessary to load the MIB file information into the CIM repository. This can be done with the MIB compiler delivered with WMI, called **SMI2SMIR.Exe**, which basically converts the Structure of Management Information (SMI) format to the Structure of Management Information Repository (SMIR) format. Simply said, this utility converts a MIB file into a MOF file loadable in the CIM repository. To locate the information you are interested in, it must be clear that the challenge is not about WMI anymore but is a pure SNMP issue. Knowing the SNMP representation of the device you want to manage is key!

A Cisco router has a lot of private MIB information. So let’s take a very easy example of private MIB information. For those of you who are not familiar with the Cisco routers, you should know that the device contains a flash memory, which contains the Cisco Internetwork Operating System (IOS). The IOS is nothing but a binary file stored in this flash memory. If

you execute a TELNET session on the Cisco router and execute the **show flash** command, you will see the IOS binary file loaded in the flash (see Figure 3.44).



```
Telnet Cisco-Brussels.LissWare.Net

User Access Verification
Password: Brussels>en
Password: Brussels#sh flash

System flash directory:
File  Length  Name/status
1    4271008  c2500-c-l_112-3p.bin
[4271072 bytes used, 4117536 available, 8388608 total]
8192K bytes of processor board System flash (Read ONLY)

Brussels#>
```

Figure 3.44 The Cisco IOS loaded in the memory flash.

As we can see in Figure 3.44, we have an IOS binary file, called “c2500-c-l_112-3p.bin,” of 4,271,008 bytes (4,171 KB) size. The flash memory card has a total size of 8,388,608 bytes (8 MB). To retrieve this information from WMI via SNMP, we must use a private Cisco MIB file. This MIB file is called **CISCO-FLASH-MIB.my** and is available from the Cisco Web site for download. To successfully convert the MIB file into a MOF file, we also need the **CISCO-SMI.my** MIB file, since it contains definitions used in **CISCO-FLASH-MIB.my**. The conversion to a MOF file can be created, as shown in Sample 3.73.

Sample 3.73 Creating a private MIB in the CIM repository

```
C:\>smi2smir /m 0 /g CISCO-FLASH-MIB.my CISCO-SMI.my > CISCO-FLASH-MIB.MOF

smi2smir : Version 1.50.1085.0000 : MIB definitions compiled from "CISCO-FLASH-MIB.my"

smi2smir : Syntax Check successful on "CISCO-FLASH-MIB.my"

smi2smir : Version 1.50.1085.0000 : MIB definitions compiled from "CISCO-SMI.my"

smi2smir : Syntax Check successful on "CISCO-SMI.my"
```

```
smi2smir : Semantic Check successful on "CISCO-FLASH-MIB.my"
smi2smir: Generated MOF successfully
```

Note the presence of the redirection “>” character on the command line. **SMI2SMIR.Exe** outputs the generated MOF on the screen. So, if you want this in a file, you must redirect the output. Table 3.70 shows the complete list of command-line parameters for **SMI2SMIR.Exe**.

Table 3.70*The SMI2SMIR.Exe Command-Line Parameters*

smi2smir [<DiagnosticArgs>] [<VersionArgs>] [<IncludeDirs>] <CommandArgs> <MIB file> [<Import Files>]	
smi2smir [<DiagnosticArgs>] <RegistryArgs> [<Directory>]	
smi2smir <ModuleInfoArgs> <MIB file>	
smi2smir <HelpArgs>	
DiagnosticArgs:	
/m <diagnostic level> Specifies the kind of diagnostics to display: 0 (silent), 1 (fatal), 2 (fatal and warning), or 3 (fatal, warning, and information messages).	
/c <count> Specifies the maximum number of fatal and warning messages to display.	
VersionArgs:	
/v1 Specifies strict conformance to the SNMPv1 SMI.	
/v2c Specifies strict conformance to the SNMPv2 SMI.	
CommandArgs:	
/d Deletes the specified module from the SMIR.	
/p Deletes all modules in the SMIR.	
/l Lists all modules in the SMIR.	
/lc Performs a local syntax check on the module.	
/ec [<CommandModifier>] Performs local and external checks on the module.	
/a [<CommandModifier>] Performs local and external checks and loads the module into the SMIR.	
/sa [<CommandModifier>] Same as /a, but works silently.	
/g [<CommandModifier>] Generates a SMIR MOF file that can be loaded later into CIMOM (using the MOF compiler). Used by the SNMP class provider to dynamically provide classes to one or more namespaces.	
/gc [<CommandModifier>] Generates a static MOF file that can be loaded later into CIMOM as static classes for a particular namespace.	
CommandModifiers:	
/ch Generates context information (date, time, host, user, etc.) in the MOF file header. Use with /g and /gc.	
/t Also generates SnmpNotification classes. Use with /a, /sa, and /g.	
/ext Also generates SnmpExtendedNotification classes. Use with /a, /sa, and /g.	
/t/o Generates only SnmpNotification classes. Use with /a, /sa, and /g.	
/ext/o Generates only SnmpExtendedNotification classes. Use with /a, /sa, and /g.	
/s Does not map the text of the DESCRIPTION clause. Use with /a, /sa, /g, and /gc.	
/auto Rebuilds the MIB lookup table before completing <CommandArg> switch. Use with /ec, /a, /g, and /gc.	
IncludeDirs:	
/i <directory> Specifies a directory to be searched for dependent MIB modules. Use with /ec, /a, /sa, /g, and /gc.	
RegistryArgs:	
/pa Adds the specified directory to the registry (Default is current directory).	
/pd Deletes the specified directory from the registry (Default is current directory).	
/pl Lists the MIB lookup directories in the registry.	
/r Rebuilds the entire MIB lookup table.	
ModuleInfoArgs:	
/n Returns the ASN.1 name of the specified module.	
/ni Returns the ASN.1 names of all imports modules referenced by the input module.	
HelpArgs:	
/h Displays this usage information.	
/? Displays this usage information.	
For auto-detection of dependent MIBs, the following values of type REG_MULTI_SZ must be set under the root key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\Providers\SNMP\Compiler: “File Path” : An ordered list of directory names where MIBs are located. “File Suffixes” : An ordered list of file extensions for MIB files.	

Once the MOF file generated, it is ready to be loaded in the CIM repository, as follows:

```
C:\>MOFCOMP.Exe CISCO-FLASH-MIB.MOF
Microsoft (R) 32-bit MOF Compiler Version 1.50.1085.0007
Copyright (c) Microsoft Corp. 1997-1999. All rights reserved.

Parsing MOF file: CISCO-FLASH-MIB.MOF
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

Note that the MOF file will be loaded in the `Root\SNMP\SMIR` namespace. Once completed, the `Root\SNMP\CiscoBrussels` namespace should show WMI classes representing the Cisco Flash SNMP information, as shown in Figure 3.45. Ensure that the Cisco router is available on the network during this operation. If the Cisco router defined in the `AgentAddress` qualifier is not reachable, it is likely that the private MIB information stored in the MOF file will be improperly displayed.

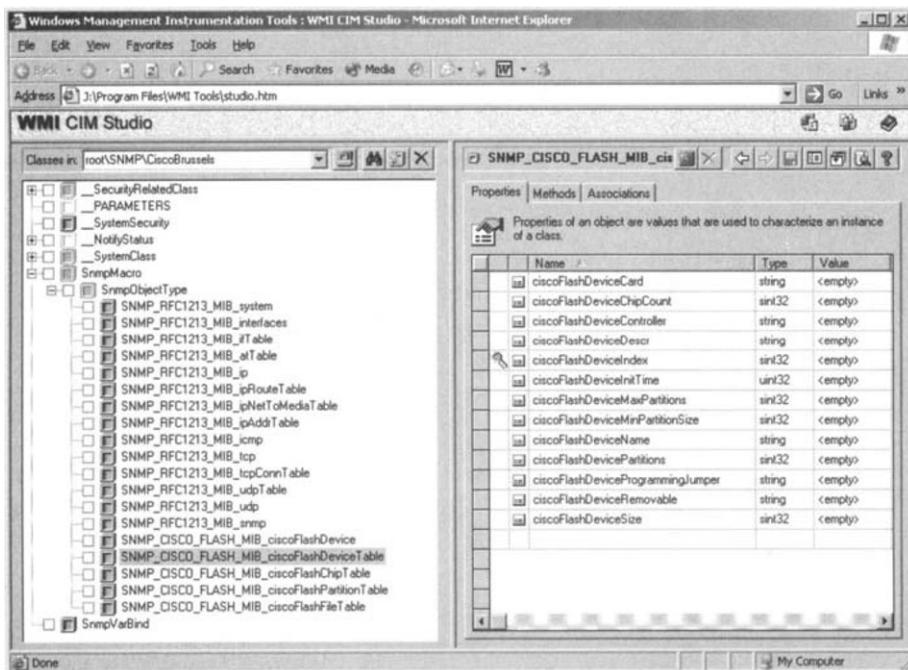


Figure 3.45 The Cisco Flash SNMP classes in the CIM repository.

If we reuse Sample 3.71 (“Obtaining remote device IP addresses from WMI via SNMP”) and change line 27:

```
27: Const cWMIClass = "SNMP_RFC1213_MIB_ipAddrTable"  
to:  
27: Const cWMIClass = "SNMP_CISCO_FLASH_MIB_ciscoFlashFileTable"  
we should obtain an output similar to this one:
```

```
1: C:\>GetInstanceWithAPI2.wsf  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5:  
6: - SNMP_CISCO_FLASH_MIB_ciscoFlashFileTable -----  
7: *ciscoFlashDeviceIndex: ..... 1  
8: ciscoFlashFileChecksum: ..... 3078424434  
9: *ciscoFlashFileIndex: ..... 1  
10: ciscoFlashFileName: ..... c2500-c-1_112-3p.bin  
11: ciscoFlashFileSize: ..... 4271008  
12: ciscoFlashFileStatus: ..... valid  
13: *ciscoFlashPartitionIndex: ..... 1
```

We recognize the Cisco IOS binary file name (line 10) with its size (line 11), because we saw it in Figure 3.45. However, we do not see the flash memory size, because another WMI class, called *SNMP_CISCO_FLASH_MIB_ciscoFlashPartitionTable*, provides this information.

If the WMI classes representing the SNMP information are not shown, it is likely due to a feature part of the *SNMP* WMI providers called the correlation. Correlation might prevent you from seeing all classes that are actually supported by a device. In such a case, it could be wise to turn the correlation OFF. The side effect of turning correlation OFF is that many classes not supported by a device may appear in the SMIR. When correlation is turned ON, the *SNMP* provider causes an enumeration of device-supported classes, similar to the result of running an SNMP MIB walk operation. This could produce a performance cost in using correlation, because the device is queried to see what classes it supports. To set the correlation OFF, the *Correlate* parameter (or qualifier, if you use a dedicated namespace) set in the *SWbemNamedValueSet* object must be set to FALSE (OFF) or TRUE (ON). This value is set in Sample 3.71, line 38.

3.7.5.3 **Organizing the SNMP data access**

As we can see, based on the fact that the SNMP device is local or remote, uses a private MIB (i.e., Cisco device), and is WMI enabled (i.e., Windows

computer), there are different options to access SNMP data. These various options can be quite confusing, since we have different ways of organizing the SNMP data in WMI and different protocols to access that information (i.e., RPC and SNMP). Let's try to summarize this with three figures showing the various options (see Figures 3.46 through 3.48).

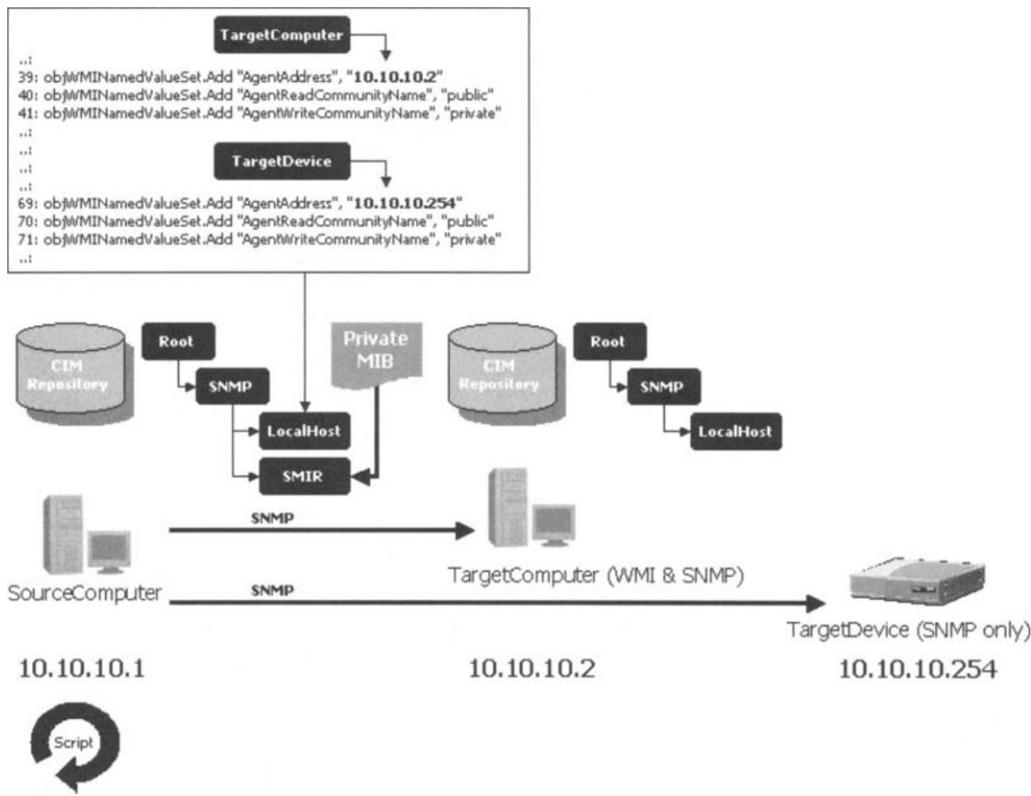


Figure 3.46 Accessing SNMP data via the localhost namespace and the *SWBemNamedValueSet* object.

The first option is the easiest one, since it consists of using the default SNMP CIM repository configuration and overriding some settings (i.e., *AgentAddress*) with an *SWBemNamedValueSet* object (see Figure 3.46) from the script. This technique gives the ability to access any SNMP device from any WMI computer in a Windows network, since no particular CIM repository customization is required. The developed script code can be run on any WMI computer. However, if some private MIB information must be accessed, it is necessary to load the related MIB information on a specific WMI system and connect the script to the WMI system where the private

MIB information is loaded. An important point to note in this scenario is that SNMP is the only protocol used to access the desired information across the network.

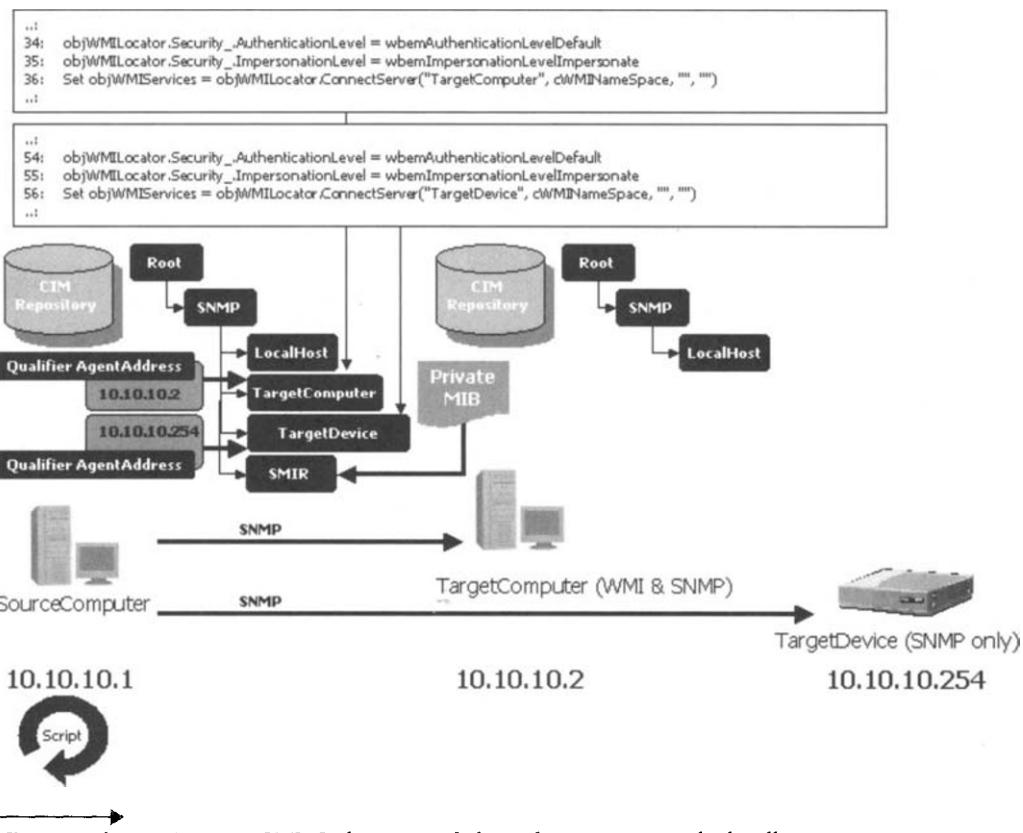


Figure 3.47 Accessing SNMP data via a dedicated namespace on the localhost.

The second technique consists of the creation of dedicated namespaces for each SNMP computer or device that must be accessed (see Figure 3.47). This technique requires a specific customization of the CIM repository and links the script to the system hosting the namespaces. Creating a dedicated namespace has the advantage of enabling the SNMP access to any tool running on top of WMI, since no specific logic is required to create an SWBemNamedValueSet object. Since all SNMP parameters are stored in the Qualifiers of the dedicated namespaces, any tool enumerating instances of the classes available in these namespaces will retrieve the SNMP information. As in the first case, the protocol used on the network is SNMP only. If some private MIB information must be accessed, the private MIB files must be loaded in the CIM repository.

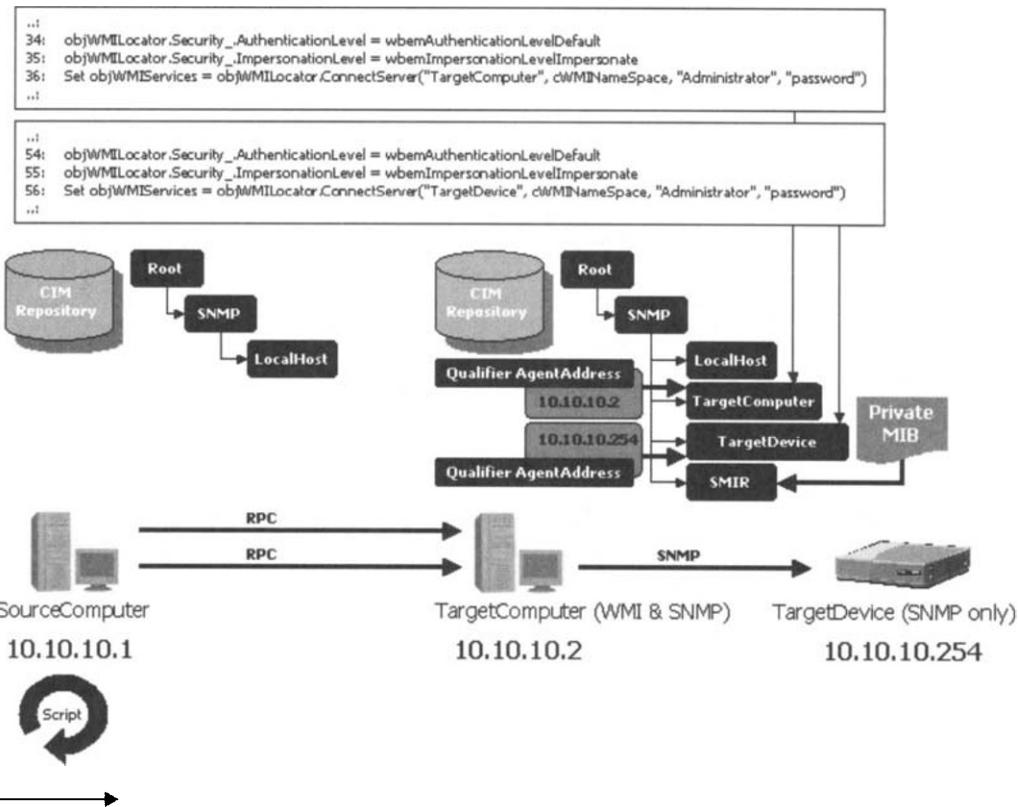


Figure 3.48 Accessing SNMP data through a remote WMI computer via a dedicated namespace.

The third technique is very similar to the second one (see Figure 3.48). However, in this case, the script code is run on a system where the dedicated namespaces are not available. Since WMI is COM/DCOM based, it is possible, by using the WMI scripting API (i.e., *ConnectServer* method of the SWBemServices object), to access a remote WMI host, which contains the dedicated namespace for the SNMP devices. Of course, as in the first technique, it is always possible to override the namespaces SNMP communication parameters in the remote hosts (Qualifiers) with an SWBemNamedValueSet object. The interesting feature in this last case is that you must authenticate to the remote WMI system first (on top of the COM/DCOM mechanisms, which are RPC based) and then access the SNMP devices, as done previously (via the SNMP protocol). If some private MIB information must be accessed, the private MIB files must be loaded in the CIM repository of the remote WMI system.

3.7.5.4 Receiving SNMP traps

With the WMI *SNMP* providers it is possible to receive SNMP traps or notifications in addition to simply retrieving SNMP information. If you remember, in the beginning of this section, we mentioned that the *SNMP* providers are also implemented as event providers. We must distinguish two event provider types:

- **SNMP Encapsulated Event provider (SEEP):** The encapsulated provider means that the event instance has simple properties describing the information mapped directly from the SNMP trap. This corresponds to the *SnmpNotification* parent class with its subclasses.
- **SNMP Referent Event provider (SREP):** The referent provider abstracts the information present within the SNMP trap so that properties that share the same class and instance are presented as embedded objects. This allows for the unique instance, with which the trap is associated, to be retrieved after the receipt of the event, using the SNMP *_RELPATH*. This corresponds to the *SnmpExtendedNotification* parent class with its subclasses.

In both cases, the same information is provided. Only the manner in which this information is presented and structured is different.

Figure 3.40 shows the event classes (left pane) supported by the *SNMP* event provider. The *SNMP* event providers support three types of traps or notifications:

- **Generic:** These traps correspond to events such as link up and cold start. These traps are listed in Table 3.68. These classes are subclasses of the *SnmpNotification* and *SnmpExtendedNotification* classes.
- **Enterprise specific:** These traps correspond to events that are represented by a WMI class that is not a subclass of the *SnmpNotification* and *SnmpExtendedNotification* classes. To support enterprise-specific traps and notifications, an event consumer must define classes in the CIM repository by compiling MIB definitions using the SNMP MIB compiler (such as that made for the Cisco device to retrieve private MIB information about the memory flash but related to traps or notifications).
- **Enterprise nonspecific:** These traps do not correspond to any of the generic event types or enterprise-specific event types. Nonenterprise-specific traps and notifications do not have their MIB definitions compiled into the SMIR. These traps are listed in Table 3.68.

We can use the Cisco router to show how to capture traps sent by the device at startup. To monitor SNMP traps, we will reuse Sample 6.17 ("A generic script for asynchronous event notification") in the appendix). This would generate the following result:

```

1:  C:\>GenericEventAsyncConsumer.wsf "Select * From SnmpColdStartNotification
   Where AgentAddress='10.10.10.253'" /Namespace:Root\SNMP\CiscoBrussel
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Saturday, 02 February, 2002 at 16:05:16: 'SnmpColdStartNotification' has been triggered.
9:
10: - SnmpColdStartNotification -----
11: AgentAddress: ..... 10.10.10.253
12: AgentTransportAddress: ..... 10.10.10.253
13: AgentTransportProtocol: ..... IP
14: Community: ..... public
15: Identification: ..... 1.3.6.1.6.3.1.1.5.1
16: TIME_CREATED: ..... 02-02-2002 15:05:16 (20020202150516.010497+060)
17: TimeStamp: ..... 1096
18:
19: END - OnObjectReady.
20:
21: BEGIN - OnObjectReady.
22: Saturday, 02 February, 2002 at 16:05:18: 'SnmpColdStartNotification' has been triggered.
23:
24: - SnmpColdStartNotification -----
25: AgentAddress: ..... 10.10.10.253
26: AgentTransportAddress: ..... 10.10.10.253
27: AgentTransportProtocol: ..... IP
28: Community: ..... public
29: Identification: ..... 1.3.6.1.6.3.1.1.5.1
30: TIME_CREATED: ..... 02-02-2002 15:05:18 (20020202150518.010497+060)
31: TimeStamp: ..... 1096
32:
33: END - OnObjectReady.
```

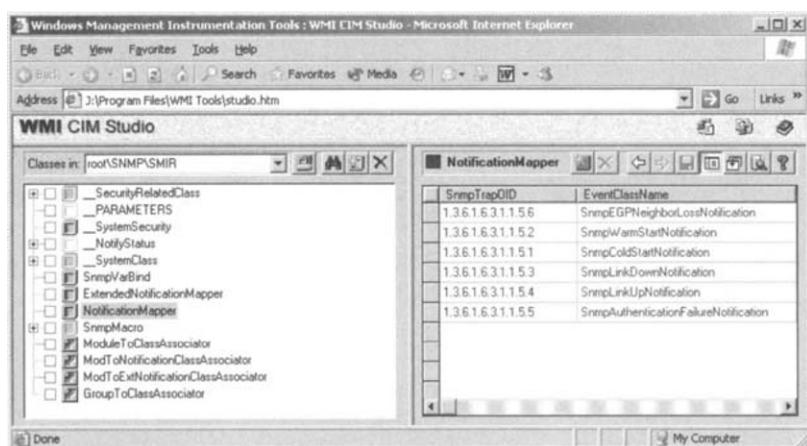
Since the purpose is to receive traps related to the device startup, the WQL event query uses the *SnmpColdStartNotification* event class. We see in the previous output sample that the Cisco device sends two traps (lines 7 through 19 and lines 21 through 33). Because the *SnmpColdStartNotification* class is a generic trap representation coming from RFC 1215, we get the strict minimum information about the Cisco device itself. The dotted number contained in the *Identification* property (line 29) is the OID number used by the SNMP trap *coldStart*.

To get the SNMP trap-specific information without searching for the corresponding Cisco MIB (enterprise-specific traps), we can use a trick. This trick is not the ideal solution (the ideal solution will be the use of the specific Cisco MIB file), but when we must deal with a device for which

very little information is available, this work-around can save a lot of time while giving a fairly usable solution. For this we must modify a definition made in the CIM repository in the `Root\SNMP\SMIR` namespace and enhance a portion of the script code.

In the `Root\SNMP\SMIR` namespace, there is a class called the `NotificationMapper` class. This class has several instances that correspond to the SNMP event standard classes. As we can see in Figure 3.49, among several instances, we have an instance with the OID number 1.3.6.1.6.3.1.1.5.1 in correspondence with the `SnmpColdStartNotification` event class.

Figure 3.49
The `NotificationMapper` instances.



If a trap is received by WMI for which there is no corresponding SNMP event class for the received OID number, then WMI triggers an extrinsic event with the `SnmpV1Notification` or `SnmpV2Notification` event class. This corresponds to an enterprise nonspecific trap. If we delete the instance using the OID number 1.3.6.1.6.3.1.1.5.1, then WMI will not be able to map the received trap to the generic `SnmpColdStartNotification` event class. In such a case, as explained, WMI will map the trap to the `SnmpV1Notification` or `SnmpV2Notification` event class. Now, if we take a closer look at the `SnmpV1Notification` or `SnmpV2Notification` classes (Figure 3.50), we see the `VarBindList` property, which is an array of `SnmpVarBind` objects.

The `SnmpVarBind` class is shown in Figure 3.51.

As we can see, this class exposes three properties, called `Encoding`, `Object-Identifier`, and `Value`, where the `Value` property is an array of integers. This is an important detail and we will see how to read this information when examining Sample 3.75.

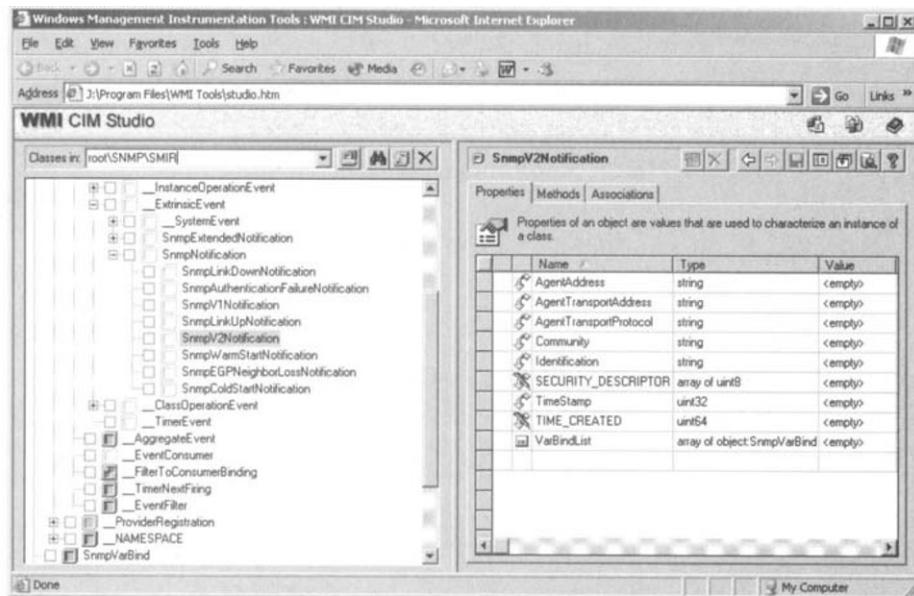


Figure 3.50 The *SnmpV2Notification* class and the *varBindList* property.

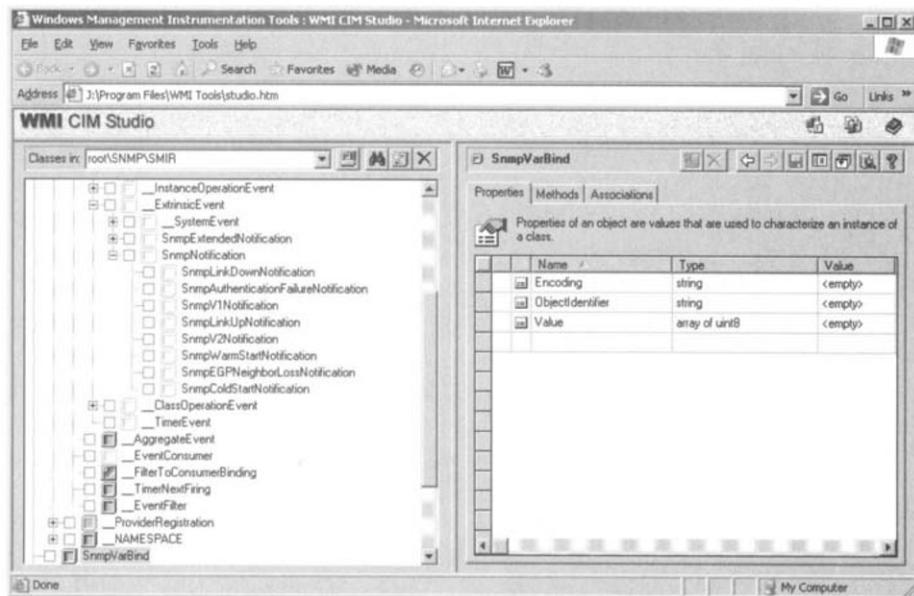


Figure 3.51 The *SnmpVarBind* class is an array of object instances.

Now that the *NotificationMapper* instance using the OID number 1.3.6.1.6.3.1.1.5.1 is deleted and before examining the code, let's see the result when the Cisco router sends the same traps as before (*coldStart*).

```
1:  C:\>GenericEventAsyncConsumer.wsf "Select * From SnmpV2Notification
2:    Where AgentAddress='10.10.10.253'" /Namespace:Root\SNMP\CiscoBrussels
3:  Microsoft (R) Windows Script Host Version 5.6
4:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
5:
6:  Waiting for events...
7:
8: BEGIN - OnObjectReady.
9: Saturday, 02 February, 2002 at 17:47:38: 'SnmpV2Notification' has been triggered.
10: AgentAddress (wbemCimtypeString) = 10.10.10.253
11: AgentTransportAddress (wbemCimtypeString) = 10.10.10.253
12: AgentTransportProtocol (wbemCimtypeString) = IP
13: Community (wbemCimtypeString) = public
14: Identification (wbemCimtypeString) = 1.3.6.1.6.3.1.1.5.1
15: SECURITY_DESCRIPTOR (wbemCimtypeUint8) = (null)
16: TIME_CREATED (wbemCimtypeUint64) = 02-02-2002 16:47:38 (20020202164738.823428+060)
17: TimeStamp (wbemCimtypeUint32) = 1096
18: 1.3.6.1.2.1.1.3.0 (TimeTicks) = 1095
19: 1.3.6.1.4.1.9.2.1.2.0 (OCTET STRING) = reload
20: 1.3.6.1.3.1057.1 (IpAddress) = 10.10.10.253
21: 1.3.6.1.6.3.1.1.4.3.0 (OBJECT IDENTIFIER) = 1.3.6.1.4.1.9.1.19
22:
23: END - OnObjectReady.
24:
25: BEGIN - OnObjectReady.
26: Saturday, 02 February, 2002 at 17:47:39: 'SnmpV2Notification' has been triggered.
27: AgentAddress (wbemCimtypeString) = 10.10.10.253
28: AgentTransportAddress (wbemCimtypeString) = 10.10.10.253
29: AgentTransportProtocol (wbemCimtypeString) = IP
30: Community (wbemCimtypeString) = public
31: Identification (wbemCimtypeString) = 1.3.6.1.6.3.1.1.5.1
32: SECURITY_DESCRIPTOR (wbemCimtypeUint8) = (null)
33: TIME_CREATED (wbemCimtypeUint64) = 02-02-2002 16:47:39 (20020202164739.013702+060)
34: TimeStamp (wbemCimtypeUint32) = 1100
35: 1.3.6.1.2.1.1.3.0 (TimeTicks) = 1099
36: 1.3.6.1.4.1.9.2.1.2.0 (OCTET STRING) = reload
37: 1.3.6.1.3.1057.1 (IpAddress) = 10.10.10.253
38: 1.3.6.1.6.3.1.1.4.3.0 (OBJECT IDENTIFIER) = 1.3.6.1.4.1.9.1.19
39: END - OnObjectReady.
```

The result now displays the information contained in the *SnmpVarBind* instances, which are themselves contained in the *varBindList* array made of a collection of instances. For example, at lines 18 and 35, we see the OID number 1.3.6.1.4.1.9.2.1.2.0. This number corresponds to specific Cisco private MIB information. If you examine the Cisco documentation, you will discover that this SNMP information shows the reason for the last device reboot. In this case, since we used the Cisco *reload* command, we see the text “reload” (lines 18 and 35). To translate Cisco MIB OID numbers

to a readable form (1.3.6.1.4.1.9.2.1.2.0 to *whyReload*), you can also use the following URL: <http://jaguar.ir.miami.edu/%7Emarcus/snmptrans.html>.

The script we use to display this information is still Sample 6.17 (“A generic script for asynchronous event notification”) in the appendix. However, at line 112 (see the following extract), this script invokes the DisplayProperties() function from the sink routine (the DisplayProperties() function is available in Sample 4.30, “A generic routine to display the SWbemPropertySet object” in the appendix).

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
10:
11:    <?job error="True" debug="True" ?>
12:
13:    <runtime>
.:
19:    </runtime>
20:
21:    <script language="VBScript" src=".\\Functions\\DisplayInstanceProperties (With SNMP).vbs" />
22:    <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
23:    <script language="VBScript" src=".\\Functions\\PauseScript.vbs" />
24:
25:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
26:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
27:
28:    <script language="VBScript">
29:      <![CDATA[
.:
33:      ' -----
34:      Const cComputerName = "LocalHost"
35:      Const cWMINameSpace = "Root\cimv2"
.:
103:      ' -----
104:      Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)
105:
106:        Wscript.Echo
107:        Wscript.Echo "BEGIN - OnObjectReady."
108:        WScript.Echo FormatDateTime(Date, vbLongDate) & " at " & _
109:                      FormatDateTime(Time, vbLongTime) & ":" "" & _
110:                      objWbemObject.Path_.Class & "' has been triggered."
111:
112:        DisplayProperties objWbemObject, 2
113:        Wscript.Echo
114:
115:        Wscript.Echo "END - OnObjectReady."
116:
117:      End Sub
118:
.:
127:
128:    ]]>
129:  </script>
130: </job>
131:</package>
```

To display the *SnmpVarBind* instances, we updated the *DisplayProperties()* function accordingly. The code is shown in Sample 3.74. The code of this updated subroutine starts as the original one (lines 14 through 30). The only change made in the *DisplayProperties()* function concerns the test of the property name (line 31) and the invocation of the *DisplaySNMPBindings()* function (line 32), if the property is the *VarBindList* property exposed by the *SnmpV1Notification* or *SnmpV2Notification* classes.

Sample 3.74 The updated *DisplayProperties()* function to display the *SnmpVarBind* instances

```

.:
6: ' -----
7:Function DisplayProperties (objWMInstance, intIndent)
.:
14:     Set objWMIPropertySet = objWMInstance.Properties_
15:     For Each objWMIProperty In objWMIPropertySet
16:
17:         boolCIMKey = objWMIProperty.Qualifiers_.Item("key").Value
18:         If Err.Number Then
19:             Err.Clear
20:             boolCIMKey = False
21:         End If
22:         If boolCIMKey Then
23:             strCIMKey = "*"
24:         Else
25:             strCIMKey = ""
26:         End If
27:
28:         If Not IsNull (objWMIProperty.Value) Then
29:             If objWMIProperty.CIMType = wbemCimtypeObject Then
30:                 If objWMIProperty.isArray Then
31:                     If objWMIProperty.Name = "VarBindList" Then
32:                         DisplaySNMPBindings objWMIProperty, intIndent
33:                     Else
34:                         For Each varElement In objWMIProperty.Value
35:                             WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
36:                                         " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ")"
37:                             DisplayProperties varElement, intIndent + 2
38:                         Next
39:                     End If
40:                 Else
41:                     WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
42:                                         " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ")"
43:                     DisplayProperties objWMIProperty.Value, intIndent + 2
44:                 End If
45:             Else
46:
47:                 End If
48:             Else
49:
50:                 End If
51:             Next
52:
53:
54:     End Function
55:
56:
57:
58:
59:
60:
61:
62:
63:End Function
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80: Next
81:
82:
83:End Function
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
```

Next, if the property matches the *varBindList* property, Sample 3.75, which contains the *DisplaySNMPBindings()* function, is executed.

Sample 3.75 *The DisplaySNMPBindings() function to display the SnmpVarBind instances*

```

...:
...:
...:
84:
85: -----
86:Function DisplaySNMPBindings (varBindList, intIndent)
...:
96:    For Each varBindElement In varBindList.Value
97:        Set objWMIPropertySet = varBindElement.Properties_
98:
99:        varSNMPValue = ""
100:
101:       For Each objWMIProperty In objWMIPropertySet
102:           Select Case objWMIProperty.Name
103:               Case "Encoding"
104:                   strSNMPEncoding = objWMIProperty.Value
105:
106:               Case "ObjectIdentifier"
107:                   strSNMPID = objWMIProperty.Value
108:
109:               Case "Value"
110:                   Select Case strSNMPEncoding
111:                       Case "TimeTicks"
112:                           varSNMPValue = objWMIProperty.Value(0) + _
113:                                         objWMIProperty.Value(1) * 256 + _
114:                                         objWMIProperty.Value(2) * 256 * 256 + _
115:                                         objWMIProperty.Value(3) * 256 * 256 * 256
116:                       Case "INTEGER"
117:                           varSNMPValue = objWMIProperty.Value(0) + _
118:                                         objWMIProperty.Value(1) * 256 + _
119:                                         objWMIProperty.Value(2) * 256 * 256 + _
120:                                         objWMIProperty.Value(3) * 256 * 256 * 256
121:                       Case "OCTET STRING"
122:                           For Each varElement In objWMIProperty.Value
123:                               varSNMPValue = varSNMPValue & Chr(varElement)
124:                           Next
125:                       Case "IpAddress"
126:                           For Each varElement In objWMIProperty.Value
127:                               If Len (varSNMPValue) = 0 Then
128:                                   varSNMPValue = varElement
129:                               Else
130:                                   varSNMPValue = varSNMPValue & "." & varElement
131:                               End If
132:                           Next
133:                       Case "OBJECT IDENTIFIER"
134:                           For intIndice=0 to Ubound (objWMIProperty.Value) Step 4
135:                               varElement = objWMIProperty.Value(0 + intIndice) + _
136:                                             objWMIProperty.Value(1 + intIndice) * 256 + _
137:                                             objWMIProperty.Value(2 + intIndice) * 256 * 256 + _
138:                                             objWMIProperty.Value(3 + intIndice) * 256 * 256 * 256
139:                           If Len (varSNMPValue) = 0 Then
140:                               varSNMPValue = varElement
141:                           Else

```

```
142:                     varSNMPValue = varSNMPValue & "." & varElement
143:                 End If
144:             Next
145:         Case Else
146:             varSNMPValue = "<unknown coding>"
147:
148:         End Select
149:
150:     Case Else
151:
152:     End Select
153: Next
154: Set objWMIPropertySet = Nothing
155:
156: WScript.Echo strSNMPOID & " (" & strSNMPEncoding & ") = " & varSNMPValue
157:
158: Next
159:
160:End Function
...:
...:
...:
```

Because the *VarBindList* property is an array, the values of this property are enumerated (lines 96 through 158). Next, since each of these values is an object made from the *SnmpVarBind* class, the script extracts the properties of the examined *SnmpVarBind* instance (line 97). As we have done many times now, each property is enumerated in a loop for further examination (lines 101 through 153). Based on the property name of the *SnmpVarBind* instance (line 102), the value of the property is retrieved accordingly (lines 103, 106, and 109). To retrieve the value of the *Value* property, some special care must be taken. Because the *SnmpVarBind* properties enumeration retrieves the encoding type of the *Value* property from the *Encoding* property (line 104), the script uses an appropriate decoding technique based on the value of the *Encoding* property (lines 110 through 148). Four encoding types are considered:

- **TIMETICKS:** The TIMETICKS encoding is a value made up of 4 bytes. Lines 112 through 115 calculate the corresponding value.
- **INTEGER:** The INTEGER encoding is an integer. Lines 117 through 120 calculate the corresponding value.
- **OCTET STRING:** The OCTET STRING is an array of bytes. Lines 112 through 124 concatenate each byte of the array in one single string.
- **IpAddress:** The IpAddress is an array of bytes. Lines 126 through 131 concatenate each byte of the array in one single dotted string to represent a well-formatted IP address.

- **OBJECT IDENTIFIER:** The OBJECT IDENTIFIER is an array of bytes, where each 4 bytes represent an integer. Lines 134 through 144 calculate the value of the integer and concatenate each value in one single dotted string to represent a well-formatted OID number.

Once the value extraction and formatting are completed, the script displays the SNMP information in a readable form (line 156).

3.7.5.5 **Sending SNMP commands**

If it is possible to read SNMP data from an SNMP-enabled device, it is also possible to write some information via SNMP. Of course, the device managed with SNMP must support a set of SNMP commands. Again, we need the SNMP knowledge and information available from the device itself. For example, with the Cisco router we used in the previous examples, it is possible to send an SNMP command to reload the device. Of course, the router must be configured properly at the SNMP level to accept such a command for obvious security reasons. Figure 3.43 (“Enabling SNMP on your Cisco router”) shows the Cisco router SNMP configuration to access the device via SNMP. However, in order to make the Cisco router accept an SNMP reload command, it is necessary to add the following statements:

```
1: snmp-server community private RW  
2: snmp-server system-shutdown
```

Line 1 is already part of the configuration shown in Figure 3.43 and defines the community string for SNMP write operations. However, line 2 is mandatory to reboot the Cisco router via SNMP. Of course, the primary goal is not to demonstrate how to reboot active network devices via SNMP but to show how, from a script written on top of WMI, it is possible to send an SNMP command to an SNMP-enabled device, where the SNMP command is defined in a private MIB of a manufacturer. The overall principle will always be the same despite the device used.

First of all, because the **Reboot** command is defined in a private MIB of Cisco, it is necessary to convert the desired MIB to a MOF file. The SNMP **reload** command definition is defined in the **OLD-CISCO-TS-MIB.My** MIB. This MIB, similar to the one we used previously, is also available from the Cisco Web site. As we did previously, the MIB file must be converted to a MOF file with the following command line:

```
C:\>smi2smir /g OLD-CISCO-TS-MIB.My CISCO-SMI.mys > OLD-CISCO-TS-MIB.Mof
```

Next, in order to make this SNMP information available to WMI, the generated MOF file must be loaded in the CIM repository with **MOF-COMP.EXE**:

```
C:\>MOFCOMP OLD-CISCO-TS-MIB.Mof
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: OLD-CISCO-TS-MIB.Mof
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

Figure 3.52 shows the WMI class result obtained from the private MIB conversion.

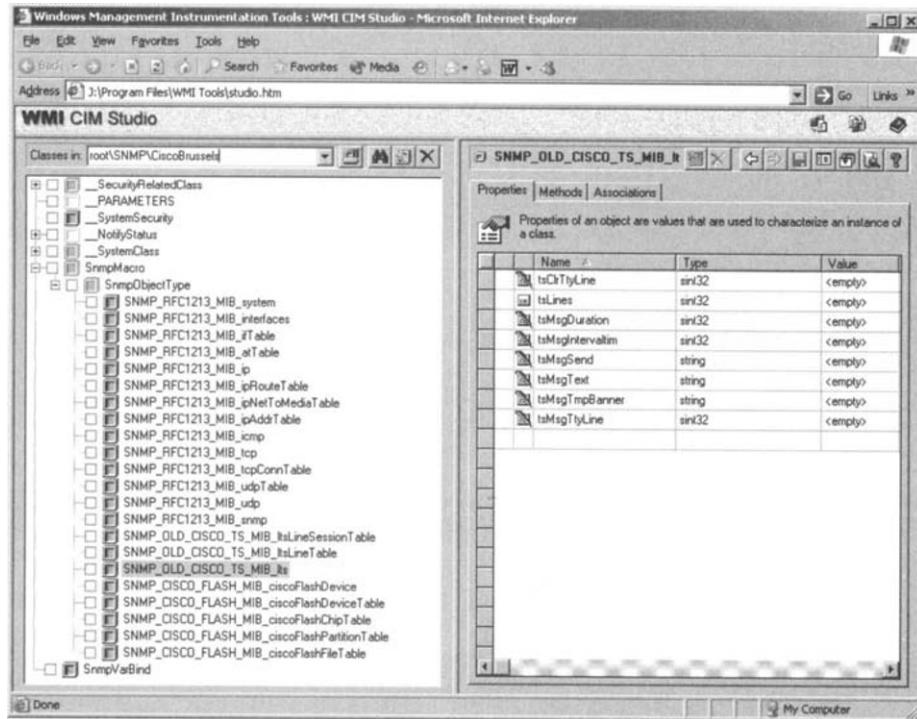


Figure 3.52 The *SNMP_OLD_CISCO_TS_MIB_Its* class to send SNMP commands to a Cisco device (SNMP-enabled).

The SNMP class to send SNMP commands to the device is called *SNMP_OLD_CISCO_TS_MIB_Its*. This class exposes a property called *tsMsgSend*. Basically, by looking at the MIB file, the *tsMsgSend* supports four parameters:

```
tsMsgSend OBJECT-TYPE
SYNTAX  INTEGER {
    nothing(1),
    reload(2),
    messagedone(3),
    abort(4)
```

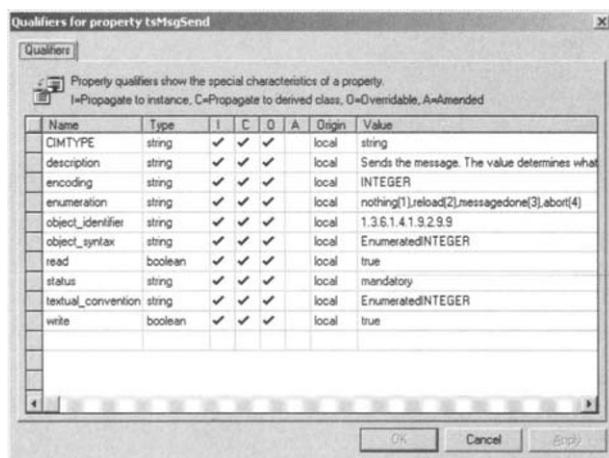
```

}
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
    "Sends the message. The value determines what
     to do after the message has completed."
::= { lts 9 }

```

These parameters are also shown in the MOF file generated from the SMI2SMIR.EXE conversion or from WMI CIM Studio by looking at the *enumeration* qualifier (see Figure 3.53). The interesting parameter for our purpose is the *Reload* parameter.

Figure 3.53
The *tsMsgSend* property qualifiers.



Of course, it must be clear that a perfect knowledge of the SNMP capabilities of the device is a requirement to find out which information to use to perform the desired management tasks. As mentioned previously, you must refer to your device manufacturer if you need support to locate the desired information.

How to proceed? In theory, the goal is simple: A script must create an instance of the *SNMP_OLD_CISCO_TS_MIB_lts* class and store the “Reload” value into the *tsMsgSend* property. Next, the script will commit the updated instance back to the CIM repository. The end result will create an SNMP write operation to the corresponding MIB property, which will reload the Cisco router. In practice, things are less simple, since some details must be taken into consideration, especially when saving the updated instance back to the CIM repository. The logic is implemented in Sample 3.76.

→ **Sample 3.76** *Sending SNMP commands*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:  <script language="VBScript" src=".\\Functions\\TinyErrorHandler.vbs" />
14:
15:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:  <object progid="WbemScripting.SWBemNamedValueSet" id="objWMINamedValueSet" />
17:
18:  <script language="VBScript">
19:    <![CDATA[
.:
23:    Const cComputerName = "LocalHost"
24:    Const cWMINameSpace = "Root/SNMP/CiscoBrussels"
25:    Const cWMIClass = "SNMP_OLD_CISCO_TS_MIB_lts"
.:
33:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
34:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
35:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
.:
38:    objWMINamedValueSet.Add "Correlate", True
39:    objWMINamedValueSet.Add "AgentAddress", "Cisco-Brussels.LissWare.Net"
40:    objWMINamedValueSet.Add "AgentFlowControlWindowSize", 3
41:    objWMINamedValueSet.Add "AgentReadCommunityName", "public"
42:    objWMINamedValueSet.Add "AgentRetryCount", 1
43:    objWMINamedValueSet.Add "AgentRetryTimeout", 500
44:    objWMINamedValueSet.Add "AgentVarBindsPerPdu", 10
45:    objWMINamedValueSet.Add "AgentWriteCommunityName", "private"
46:
47:    Set objWMISNMPInstance = objWMIServices.Get(cWMIClass & "=@")
.:
50:    objWMISNMPInstance.tsMsgSend = "Reload"
51:
52:    objWMINamedValueSet.Add "__PUT_EXTENSIONS", True
53:    objWMINamedValueSet.Add "__PUT_EXT_CLIENT_REQUEST", True
54:    objWMINamedValueSet.Add "__PUT_EXT_PROPERTIES", Array ("tsMsgSend")
55:
56:    objWMISNMPInstance.Put_ wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete, _
57:                           objWMINamedValueSet
58:    If Err.Number Then
59:      WScript.Echo "0x" & Hex(Err.Number) & " - " & Err.Description & vbCRLF
60:
61:      Set objWMILastError = CreateObject("wbemschema.swbemlasterror")
62:      WScript.Echo "Description: " & objWMILastError.Description
63:      WScript.Echo "Operation: " & objWMILastError.Operation
64:      WScript.Echo "ParameterInfo: " & objWMILastError.ParameterInfo
65:      WScript.Echo "ProviderName: " & objWMILastError.ProviderName
66:      WScript.Echo "SnmpStatusCode: 0x" & Hex (objWMILastError.SnmpStatusCode)
67:      WScript.Echo "StatusCode: 0x" & Hex(objWMILastError.StatusCode)
68:      WScript.Echo "Operation: " & objWMILastError.Operation
69:      WScript.Echo "ParameterInfo: " & objWMILastError.ParameterInfo
70:      WScript.Echo "ProviderName: " & objWMILastError.ProviderName
.:
73:      WScript.Quit (1)
74:  End If
```

```
75: 
76:     WScript.Echo "Cisco router successfully reloaded."
...:
81:     ]]>
82:     </script>
83:   </job>
84:</package>
```

From line 33 through 35, Sample 3.76 executes the WMI connection to the CIM repository. Next, from line 38 through 45, it sets up an `SWBemNamedValueSet` object defining the SNMP connection parameters (as shown in Sample 3.71). Note the assignment of the `AgentWriteCommunityName` property (line 45) with a valid community string to perform an SNMP write operation. Next, the `SNMP_OLD_CISCO_TS_MIB_Its` instance is created (line 47) and the “Reload” value is assigned to the `tsMsgSend` property (line 50). The important and interesting point concerns the scripting technique used to save the updated instance back to the CIM repository (lines 52 through 57). This technique uses the partial-instance update technique previously used in section 3.6.1.1, “Creating and updating objects in Active Directory.” It is mandatory to commit the change in this way to the CIM repository, because only the `tsMsgSend` property must be written. Note that trying to save the complete instance to the CIM repository will make the `Put_` method invocation fail. If an SNMP error occurs, it is possible to retrieve some extra information by using the `SWbemLastError` object, discussed in Chapter 5 of the first book, *Understanding WMI Scripting*. For example, if the statement “snmp-server system-shutdown” is omitted in the Cisco configuration, the error returned will be as follows:

```
1:  C:\>ReloadCisco.Wsf
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  0x80041001 - Generic failure
6:
7:  Description: Agent reported Bad Value for property 'tsMsgSend'
8:  Operation: PutInstance
9:  ParameterInfo:
10: ProviderName: MS_SNMP_INSTANCE_PROVIDER
11: SnmpStatusCode: 0x80041030
12: StatusCode: 0x80041001
13: Operation: PutInstance
14: ParameterInfo:
15: ProviderName: MS_SNMP_INSTANCE_PROVIDER
```

If everything is properly set up, the SNMP command will be sent to the Cisco device, which will be reloaded. Note that the Cisco router sends an SNMP trap after receiving this SNMP command. It is always possible to

enhance Sample 3.76 to capture the SNMP trap message confirming the successful reload of the router.

3.7.5.6 Debugging SNMP providers

In the previous example we saw that the **SWbemLastError** object provides some extra information about the last WMI error. However, when working with SNMP devices, it is not always easy to troubleshoot the installation to determine the cause of the problem. A nice feature coming with the WMI SNMP implementation is the ability to trace the *SNMP* provider activity. Basically, the tracing activity setup is the same as the *Active Directory* providers' tracing activity. It uses the same registry key values (see Table 3.53, “Enabling the Trace Logging of a WMI Provider”) but in a different registry hive:

`HKLM\SOFTWARE\Microsoft\WBEM\PROVIDERS\Logging\WBEMSNMP`

The value to use for the *Level* registry key takes an integer value from 0 through $2^{32} - 1$. The value is a logical mask consisting of 32 bits. Table 3.71 shows the bit masks to define the type of debugging output to be generated. By default, the information is contained in the **SWBEM-SNMP.LOG** file.

Table 3.71 The SNMP Debugging Output Level

Bits	Description
0	SNMP class provider SWbemServices object method invocations
1	SNMP class provider implementation
2	SNMP instance provider SWbemServices object method invocations
3	SNMP instance provider implementation
4	SNMP class library
5	SNMP SMIR
6	SNMP correlator
7	SNMP type mapping code
8	SNMP threading code
9	SNMP event provider interfaces and implementation

With a *Level* value of 35 when executing Sample 3.76 (“Sending SNMP commands”), the resulting trace is as follows:

```

1: CImpClasProv::GetObjectAsync ( (SNMP_OLD_CISCO_TS_MIB_lts) )
2: SnmpClassGetAsyncEventObject :: SnmpClassGetAsyncEventObject ()
3: SnmpClassGetAsyncEventObject :: Process ()
4: SnmpClassGetEventObject :: ProcessClass ( WbemSnmpErrorObject &a_errorObject )
5: SnmpClassGetAsyncEventObject :: ReceiveClass ( IWbemClassObject *classObject )
6: Returning from SnmpClassGetEventObject :: ProcessClass ( WbemSnmpErrorObject &a_errorObject (0) )
7: SnmpClassGetAsyncEventObject :: ReceiveComplete ()
8: Reaping Task
9: Deleting (this)
10: Returning from SnmpClassGetAsyncEventObject :: Receive4 ()
11: Returning from SnmpClassGetAsyncEventObject :: Process ()
12: WbemSnmpErrorObject :: SetMessage ( () )

```

```

13: SnmpClassGetAsyncEventObject :: ~SnmpClassGetAsyncEventObject ()
14: Sending Status
15: Returning from SnmpClassGetAsyncEventObject :: ~SnmpClassGetAsyncEventObject ()
16: Returning from CImpClasProv::GetObjectAsync ( (SNMP_OLD_CISCO_TS_MIB_ItS) ) with Result = (0)

```

3.8 Performance providers

3.8.1 High-performance providers

High-performance providers are implemented as instance providers. However, without going into the implementation details of *High-performance* providers, the major difference from standard instance providers is that they run as an in-process component of WMI or an application. (See Table 3.72.) The aim of a *High-performance* provider is to monitor data sent to the System Monitor of Windows 2000 or later. Having a *High-performance* provider has the advantage of not having WMI call the *Performance Counter* provider, which in turn, calls the performance library to collect the data. Basically, *High-performance* providers remove that layer, which greatly improves performance.

Table 3.72

The High-Performance Providers 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
High Performance Providers																
HiPerCooker_v1	Root/WMI	X						X	X			X	X			
HiPerCooker_v1	Root/CIMv2	X						X	X			X	X			
NT5_GenericPerfProvider_V1	Root/CIMv2	X						X	X			X	X	X	X	

We must distinguish between two different *High-performance* providers in Windows:

- The *Performance Counter* provider, which provides access to the raw counters data. This provider is only available in Windows 2000 and later. Its name is *NT5_GenericPerfProvider_V1*. It is available in the *Root\CIMv2* namespace. The *Performance Counter* provider supports a set of classes derived from the *Win32_PerfRawData* class (see Table 3.73).
- The *Cooked Counter* provider, which provides calculated counter data. This provider is only available under Windows XP or Windows

Table 3.73 *The Win32_PerfRawData Classes*

Name	Comments
Win32_PerfRawData_PerfProc_JobObjectDetails	% Job object Details shows detailed performance information about the active processes that make up a Job object.
Win32_PerfRawData_AspNetApplications	ASP.NET Application performance counters
Win32_PerfRawData_AspNet_AspNet	ASP.NET global performance counters
Win32_PerfRawData_AspNet_10321514_AspNetAppsv10321514	ASP.NET v1.0.3215.14 application performance counters
Win32_PerfRawData_AspNet_10321514_AspNetNetv10321514	ASP.NET v1.0.3215.14 global performance counters
Win32_PerfRawData_NTDS_NTDS	CIM_StatisticalInformation is a root class for any arbitrary collection of statistical data and/or metrics applicable to one or more managed system elements.
Win32_PerfRawData_ESE_Database	Database provides performance statistics for each process using the ESE high-performance embedded database management system.
Win32_PerfRawData_Spooler_PrintQueue	Displays performance statistics about a Print Queue.
Win32_PerfRawData_FileReplicaConn_FileReplicaConn	Displays Performance statistics of the REPLICACONN Object.
Win32_PerfRawData_MSExchangeMTA_MSExchangeMTAConnections	Each instance describes a single known entity.
Win32_PerfRawData_ESE_DatabaseInstances	Instances in this process
Win32_PerfRawData_MSDTC_DistributedTransactionCoordinator	Microsoft Distributed Transaction Coordinator performance counters
Win32_PerfRawData_MSExchangeAL_MSExchangeAL	Microsoft Exchange Address List Service
Win32_PerfRawData_MSExchangeIS_MSExchangeIS	Microsoft Exchange Information Store performance data
Win32_PerfRawData_MSExchangeIS_MSExchangeISMailbox	Microsoft Exchange Mailbox Store performance data
Win32_PerfRawData_MSExchangeIS_MSExchangeISPublic	Microsoft Exchange Public Folder Store performance data
Win32_PerfRawData_MSExchangeSA_MSExchangeSANSPIPProxy	NSPI Proxy used by Microsoft Exchange System Attendant.
Win32_PerfRawData_MSExchangeMTA_MSExchangeMTA	Only one instance : global MTA performance data.
Win32_PerfRawData_PerfProc_JobObject	Reports the accounting and processor usage data collected by each active named Job object.
Win32_PerfRawData_RSVP_RSVPInterfaces	RSVP Interfaces performance counters.
Win32_PerfRawData_RSVP_RSVPService	RSVP service performance counters.
Win32_PerfRawData_TermService_TerminalServicesSession	Terminal Services per-session resource monitoring.
Win32_PerfRawData_TermService_TerminalServices	Terminal Services summary information.
Win32_PerfRawData_ASP_ActiveServerPages	The Active Server Pages Object Type handles the Active Server Pages device on your system.
Win32_PerfRawData_PerfNet_Browser	The Browser performance object consists of counters that measure the rates of announcements, enumerations, and other Browser transmissions.
Win32_PerfRawData_PerfOS_Cache	The Cache performance object consists of counters that monitor the file system cache, an area of physical memory that stores recently used data as long as possible to permit access to the data without having to read from the disk. Because applications typically use the cache, the cache is monitored as an indicator of application I/O operations. When memory is plentiful, the cache can grow, but when memory is scarce, the cache can become too small to be effective.
Win32_PerfRawData_SMTPSVC_SMTPServer	The counters specific to the SMTP Server.
Win32_PerfRawData_PerfProc_FullImage_Costly	The Full Image performance object consists of counters that monitor the virtual address usage of images executed by processes on the computer. Full Image counters are the same counters as contained in Image object with the only difference being the instance name. In the Full Image object, the instance name includes the full file path name of the loaded modules, while in the Image object only the filename is displayed.
Win32_PerfRawData_Tcpip_ICMP	The ICMP performance object consists of counters that measure the rates at which messages are sent and received by using ICMP protocols. It also includes counters that monitor ICMP protocol errors.
Win32_PerfRawData_PerfProc_Image_Costly	The Image performance object consists of counters that monitor the virtual address usage of images executed by processes on the computer.
Win32_PerfRawData_InetInfo_InformationServicesGlobal	The Internet Information Services Global object includes counters that monitor Internet Information Services (the Web service and the FTP service) as a whole.
Win32_PerfRawData_Tcpip_IP	The IP performance object consists of counters that measure the rates at which IP datagrams are sent and received by using IP protocols. It also includes counters that monitor IP protocol errors.
Win32_PerfRawData_PerfDisk_LogicalDisk	The Logical Disk performance object consists of counters that monitor logical partitions of a hard or fixed disk drives. Performance Monitor identifies logical disks by their a drive letter, such as C.
Win32_PerfRawData_PerfOS_Memory	The Memory performance object consists of counters that describe the behavior of physical and virtual memory on the computer. Physical memory is the amount of random access memory on the computer. Virtual memory consists of the space in physical memory and on disk. Many of the memory counters monitor paging, which is the movement of pages of code and data between disk and physical memory. Excessive paging, a symptom of a memory shortage, can cause delays which interfere with all system processes.
Win32_PerfRawData_MSExchangeSRS_MSExchangeSRS	The MSExchangeSRS Object Type handles the Microsoft Exchange Site Replication service on your system.
Win32_PerfRawData_Tcpip_NBTConnection	The NBT Connection performance object consists of counters that measure the rates at which bytes are sent and received over the NBT connection between the local computer and a remote computer. The connection is identified by the name of the remote computer.

Table 3.73 The Win32_PerfRawData Classes (continued)

Name	Comments
Win32_PerfRawData_Tcpip_NetworkInterface	The Network Interface performance object consists of counters that measure the rates at which bytes and packets are sent and received over a TCP/IP network connection. It includes counters that monitor connection errors.
Win32_PerfRawData_NntpSvc_NNTPCommands	The NNTP Commands object includes counters for all NNTP commands processed by the NNTP service.
Win32_PerfRawData_NntpSvc_NNTPServer	The NNTP Server object type includes counters specific to the NNTP Server service.
Win32_PerfRawData_PerfOS_Objects	The Object performance object consists of counters that monitor logical objects in the system, such as processes, threads, mutexes, and semaphores. This information can be used to detect the unnecessary consumption of computer resources. Each object requires memory to store basic information about the object.
Win32_PerfRawData_PerfOS_PagingFile	The Paging File performance object consists of counters that monitor the paging file(s) on the computer. The paging file is a reserved space on disk that backs up committed physical memory on the computer.
Win32_PerfRawData_PerfDisk_PhysicalDisk	The Physical Disk performance object consists of counters that monitor hard or fixed disk drive on a computer. Disks are used to store file, program, and paging data and are read to retrieve these items, and written to record changes to them. The values of physical disk counters are sums of the values of the logical disks (or partitions) into which they are divided.
Win32_PerfRawData_PerfProc_ProcessAddressSpace_Costly	The Process Address Space performance object consists of counters that monitor memory allocation and use for a selected process.
Win32_PerfRawData_PerfProc_Process	The Process performance object consists of counters that monitor running application program and system processes. All the threads in a process share the same address space and have access to the same data.
Win32_PerfRawData_PerfOS_Processor	The Processor performance object consists of counters that measure aspects of processor activity. The processor is the part of the computer that performs arithmetic and logical computations, initiates operations on peripherals, and runs the threads of processes. A computer can have multiple processors. The processor object represents each processor as an instance of the object.
Win32_PerfRawData_RemoteAccess_RASTotal	The RAS performance object consists of counters that combine values for all ports of the Remote Access Service (RAS) device on the computer.
Win32_PerfRawData_RemoteAccess_RASPort	The RAS performance object consists of counters that monitor individual Remote Access Service ports of the RAS device on the computer.
Win32_PerfRawData_PerfNet_Redirector	The Redirector performance object consists of counters that monitor network connections originating at the local computer.
Win32_PerfRawData_PerfNet_Server	The Server performance object consists of counters that measure communication between the local computer and the network.
Win32_PerfRawData_PerfNet_ServerWorkQueues	The Server Work Queues performance object consists of counters that monitor the length of the queues and objects in the queues.
Win32_PerfRawData_PerfOS_System	The System performance object consists of counters that apply to more than one instance of a component processors on the computer.
Win32_PerfRawData_Tcpip_TCP	The TCP performance object consists of counters that measure the rates at which TCP Segments are sent and received by using the TCP protocol. It includes counters that monitor the number of TCP connections in each TCP connection state.
Win32_PerfRawData_TapiSrv_Telephony	The Telephony System
Win32_PerfRawData_PerfProc_ThreadDetails_Costly	The Thread Details performance object consists of counters that measure aspects of thread behavior that are difficult or time-consuming to collect. These counters are distinguished from those in the Thread object by their high overhead.
Win32_PerfRawData_PerfProc_Thread	The Thread performance object consists of counters that measure aspects of thread behavior. A thread is the basic object that executes instructions on a processor. All running processes have at least one thread.
Win32_PerfRawData_Tcpip_UDP	The UDP performance object consists of counters that measure the rates at which UDP datagrams are sent and received by using the UDP protocol. It includes counters that monitor UDP protocol errors.
Win32_PerfRawData_W3SVC_WebServiceCache	The Web Service Cache Counters object includes cache counters specific to the World Wide Web Publishing Service.
Win32_PerfRawData_W3SVC_WebService	The Web Service object includes counters specific to the World Wide Web Publishing Service.
Win32_PerfRawData_NTFSDRV_SMTPNTFSStoreDriver	This object represents global counters for the Exchange NTFS Store driver.

Server 2003. Its name is *HiPerfCooker_v1*. It is available in both the **Root\CMV2** namespace and the **Root\WMI** namespace. The *Cooked Counter* provider supports a set of classes derived from the **Win32_PerfFormattedData** class (see Table 3.74).

Table 3.74 The Win32_PerfFormattedData Classes

Name	Win32_PerfRawData class	Comments
Win32_PerfFormattedData_ESE_Database	Win32_PerfRawData_ESE_Database	Database provides performance statistics for each process using the ESE high performance embedded database management system.
Win32_PerfFormattedData_ESE_DatabaseInstances	Win32_PerfRawData_ESE_DatabaseInstances	Instances in this process
Win32_PerfFormattedData_FileReplicaConn_FileReplicaConn	Win32_PerfRawData_FileReplicaConn_FileReplicaConn	Displays Performance statistics of the REPLICACONN Object.
Win32_PerfFormattedData_MSDTC_DistributedTransactionCoordinator	Win32_PerfRawData_MSDTC_DistributedTransactionCoordinator	Microsoft Distributed Transaction Coordinator performance counters
Win32_PerfFormattedData_MSExchangeAL_MSExchangeAL	Win32_PerfRawData_MSExchangeAL_MSExchangeAL	Microsoft Exchange Address List Service
Win32_PerfFormattedData_MSExchangeIS_MSExchangeIS	Win32_PerfRawData_MSExchangeIS_MSExchangeIS	Microsoft Exchange Information Store performance data
Win32_PerfFormattedData_MSExchangeIS_MSExchangeISMailbox	Win32_PerfRawData_MSExchangeIS_MSExchangeISMailbox	Microsoft Exchange Mailbox Store performance data
Win32_PerfFormattedData_MSExchangeIS_MSExchangeISPublic	Win32_PerfRawData_MSExchangeIS_MSExchangeISPublic	Microsoft Exchange Public Folder Store performance data
Win32_PerfFormattedData_MSExchangeMTA_MSExchangeMTA	Win32_PerfRawData_MSExchangeMTA_MSExchangeMTA	Only one Instance : global MTA performance data.
Win32_PerfFormattedData_MSExchangeMTA_MSExchangeMTAConnections	Win32_PerfRawData_MSExchangeMTA_MSExchangeMTAConnections	Each instance describes a single known entity.
Win32_PerfFormattedData_MSExchangeSA_MSExchangeSANSPIProxy	Win32_PerfRawData_MSExchangeSA_MSExchangeSANSPIProxy	NSPI Proxy used by Microsoft Exchange System Attendant
Win32_PerfFormattedData_MSExchangeSRSS_MSExchangeSRSS	Win32_PerfRawData_MSExchangeSRSS_MSExchangeSRSS	The MSExchangeSRSS Object Type handles the Microsoft Exchange Site Replication service on your system.
Win32_PerfFormattedData_NTDS_NTDS	Win32_PerfRawData_NTDS_NTDS	CIM_StatisticalInformation is a root class for any arbitrary collection of statistical data and/or metrics applicable to one or more managed system elements.
Win32_PerfFormattedData_NTFSDRV_SMTPNFTSStoreDriver	Win32_PerfRawData_NTFSDRV_SMTPNFTSStoreDriver	This object represents global counters for the Exchange NTFS Store driver
Win32_PerfFormattedData_PerfDisk_LogicalDisk	Win32_PerfRawData_PerfDisk_LogicalDisk	The Logical Disk performance object consists of counters that monitor logical partitions of a hard or fixed disk drives. Performance Monitor identifies logical disks by their drive letter, such as C.
Win32_PerfFormattedData_PerfDisk_PhysicalDisk	Win32_PerfRawData_PerfDisk_PhysicalDisk	The Physical Disk performance object consists of counters that monitor hard or fixed disk drive on a computer. Disks are used to store file, program, and paging data and are read to retrieve these items, and written to record changes to them. The values of physical disk counters are sums of the values of the logical disks (or partitions) into which they are divided.
Win32_PerfFormattedData_PerfNet_Browser	Win32_PerfRawData_PerfNet_Browser	The Browser performance object consists of counters that measure the rates of announcements, enumerations, and other Browser transmissions.
Win32_PerfFormattedData_PerfNet_Redirector	Win32_PerfRawData_PerfNet_Redirector	The Redirector performance object consists of counter that monitor network connections originating at the local computer.
Win32_PerfFormattedData_PerfNet_Server	Win32_PerfRawData_PerfNet_Server	The Server performance object consists of counters that measure communication between the local computer and the network.
Win32_PerfFormattedData_PerfNet_ServerWorkQueues	Win32_PerfRawData_PerfNet_ServerWorkQueues	The Server Work Queues performance object consists of counters that monitor the length of the queues and objects in the queues.
Win32_PerfFormattedData_PerfOS_Cache	Win32_PerfRawData_PerfOS_Cache	The Cache performance object consists of counters that monitor the file system cache, an area of physical memory that stores recently used data as long as possible to permit access to the data without having to read from the disk. Because applications typically use the cache, the cache is monitored as an indicator of application I/O operations. When memory is plentiful, the cache can grow, but when memory is scarce, the cache can become too small to be effective.
Win32_PerfFormattedData_PerfOS_Memory	Win32_PerfRawData_PerfOS_Memory	The Memory performance object consists of counters that describe the behavior of physical and virtual memory on the computer. Physical memory is the amount of random access memory on the computer. Virtual memory consists of the space in physical memory and on disk. Many of the memory counters monitor paging, which is the movement of pages of code and data between disk and physical memory. Excessive paging, a symptom of a memory shortage, can cause delays which interfere with all system processes.

Table 3.74 The Win32_PerfFormattedData Classes (continued)

Name	Win32_PerfRawData class	Comments
Win32_PerfFormattedData_PerfOS_Objects	Win32_PerfRawData_PerfOS_Objects	The Object performance object consists of counters that monitor logical objects in the system, such as processes, threads, mutexes, and semaphores. This information can be used to detect the unnecessary consumption of computer resources. Each object requires memory to store basic information about the object.
Win32_PerfFormattedData_PerfOS_PagingFile	Win32_PerfRawData_PerfOS_PagingFile	The Paging File performance object consists of counters that monitor the paging file(s) on the computer. The paging file is a reserved space on disk that backs up committed physical memory on the computer.
Win32_PerfFormattedData_PerfOS_Processor	Win32_PerfRawData_PerfOS_Processor	The Processor performance object consists of counters that measure aspects of processor activity. The processor is the part of the computer that performs arithmetic and logical computations, initiates operations on peripherals, and runs the threads of processes. A computer can have multiple processors. The processor object represents each processor as an instance of the object.
Win32_PerfFormattedData_PerfOS_System	Win32_PerfRawData_PerfOS_System	The System performance object consists of counters that apply to more than one instance of a component processors on the computer.
Win32_PerfFormattedData_PerfProc_FullImage_Costly	Win32_PerfRawData_PerfProc_FullImage_Costly	The Full Image performance object consists of counters that monitor the virtual address usage of images executed by processes on the computer. Full Image counters are the same counters as contained in Image object with the only difference being the instance name. In the Full Image object, the instance name includes the full file path name of the loaded modules, while in the Image object only the filename is displayed.
Win32_PerfFormattedData_PerfProc_Image_Costly	Win32_PerfRawData_PerfProc_Image_Costly	The Image performance object consists of counters that monitor the virtual address usage of images executed by processes on the computer.
Win32_PerfFormattedData_PerfProc_JobObject	Win32_PerfRawData_PerfProc_JobObject	Reports the accounting and processor usage data collected by each active named Job object.
Win32_PerfFormattedData_PerfProc_JobObjectDetails	Win32_PerfRawData_PerfProc_JobObjectDetails	% Job object Details shows detailed performance information about the active processes that make up a Job object.
Win32_PerfFormattedData_PerfProc_Process	Win32_PerfRawData_PerfProc_Process	The Process performance object consists of counters that monitor running application program and system processes. All the threads in a process share the same address space and have access to the same data.
Win32_PerfFormattedData_PerfProc_ProcessAddressSpace_Costly	Win32_PerfRawData_PerfProc_ProcessAddressSpace_Costly	The Process Address Space performance object consists of counters that monitor memory allocation and use for a selected process.
Win32_PerfFormattedData_PerfProc_Thread	Win32_PerfRawData_PerfProc_Thread	The Thread performance object consists of counters that measure aspects of thread behavior. A thread is the basic object that executes instructions on a processor. All running processes have at least one thread.
Win32_PerfFormattedData_PerfProc_ThreadDetails_Costly	Win32_PerfRawData_PerfProc_ThreadDetails_Costly	The Thread Details performance object consists of counters that measure aspects of thread behavior that are difficult or time-consuming to collect. These counters are distinguished from those in the Thread object by their high overhead.
Win32_PerfFormattedData_RemoteAccess_RASPort	Win32_PerfRawData_RemoteAccess_RASPort	The RAS performance object consists of counters that monitor individual Remote Access Service ports of the RAS device on the computer.
Win32_PerfFormattedData_RemoteAccess_RASTotal	Win32_PerfRawData_RemoteAccess_RASTotal	The RAS performance object consists of counters that combine values for all ports of the Remote Access service (RAS) device on the computer.
Win32_PerfFormattedData_RSVP_RSVPInterfaces	Win32_PerfRawData_RSVP_RSVPInterfaces	RSVP Interfaces performance counters.
Win32_PerfFormattedData_RSVP_RSVPService	Win32_PerfRawData_RSVP_RSVPService	RSVP service performance counters.
Win32_PerfFormattedData_Spooler_PrintQueue	Win32_PerfRawData_Spooler_PrintQueue	Displays performance statistics about a Print Queue.
Win32_PerfFormattedData_TapiSrv_Telephony	Win32_PerfRawData_TapiSrv_Telephony	The Telephony System

Table 3.74 The Win32_PerfFormattedData Classes (continued)

Name	Win32_PerfRawData class	Comments
Win32_PerfFormattedData_Tcpip_ICMP	Win32_PerfRawData_Tcpip_ICMP	The ICMP performance object consists of counters that measure the rates at which messages are sent and received by using ICMP protocols. It also includes counters that monitor ICMP protocol errors.
Win32_PerfFormattedData_Tcpip_IP	Win32_PerfRawData_Tcpip_IP	The IP performance object consists of counters that measure the rates at which IP datagrams are sent and received by using IP protocols. It also includes counters that monitor IP protocol errors.
Win32_PerfFormattedData_Tcpip_NBTConnection	Win32_PerfRawData_Tcpip_NBTConnection	The NBT Connection performance object consists of counters that measure the rates at which bytes are sent and received over the NBT connection between the local computer and a remote computer. The connection is identified by the name of the remote computer.
Win32_PerfFormattedData_Tcpip_NetworkInterface	Win32_PerfRawData_Tcpip_NetworkInterface	The Network Interface performance object consists of counters that measure the rates at which bytes and packets are sent and received over a TCP/IP network connection. It includes counters that monitor connection errors.
Win32_PerfFormattedData_Tcpip_TCP	Win32_PerfRawData_Tcpip_TCP	The TCP performance object consists of counters that measure the rates at which TCP Segments are sent and received by using the TCP protocol. It includes counters that monitor the number of TCP connections in each TCP connection state.
Win32_PerfFormattedData_Tcpip_UDP	Win32_PerfRawData_Tcpip_UDP	The UDP performance object consists of counters that measure the rates at which UDP datagrams are sent and received by using the UDP protocol. It includes counters that monitor UDP protocol errors.
Win32_PerfFormattedData_TermService_TerminalServices	Win32_PerfRawData_TermService_TerminalServices	Terminal Services summary information.
Win32_PerfFormattedData_TermService_TerminalServicesSession	Win32_PerfRawData_TermService_TerminalServicesSession	Terminal Services per-session resource monitoring.

Both class types (raw and cooked) follow the same naming convention, which is of the form:

- *Win32_PerfRawData_<service_name>_<object_name>*
- *Win32_PerfFormattedData_<service_name>_<object_name>*

When dealing with TCP/IP information at the level of the network interface, we will have a class name such as *Win32_PerfRawData_Tcpip_NetworkInterface* for the raw data counters and *Win32_PerfFormattedData_Tcpip_NetworkInterface* for the cooked data counters.

The WMI AutoDiscovery/AutoPurge (ADAP) process transfers performance counter objects registered in the performance counter libraries into *Win32_PerfRawData* and *Win32_PerfFormattedData* classes in the WMI repository. When a new performance library is found, ADAP then adds a related performance object to the CIM repository. When an application is deinstalled, the deleted performance library is also detected by ADAP and it updates the CIM repository accordingly. So, basically, the ADAP process parses the performance libraries to build the *Win32_PerfRawData* and *Win32_PerfFormattedData* classes in the CIM repository.

You can look for errors generated by ADAP in two places: the NT Application Event Log (with the Event Viewer) and the WMI application event logs (in Windows\System32\Wbem\Logs\Wmiadap.log).

With the *_ADAPStatus* system class, it is possible to determine the status of the last ADAP execution. This system class exposes three read-only properties, described in Table 3.75. The *_ADAPStatus* system class is a singleton class located in the *Root\Default* namespace. The information exposed by the instance of the *_ADAPStatus* system class provides data about the most recent run of ADAP.

Table 3.75

The _ADAPStatus System Class Properties

Name	Type	Comments
<i>LastStartTime</i>	datetime	Last time the ADAP process started.
<i>LastStopTime</i>	datetime	Last time the ADAP process completed.
Status	uint32	Defines the current state of the ADAP process. 0 Process has never run on this computer. 1 Process is currently running. 2 Process is processing a performance library. 3 Process is updating and committing changes to WMI. 4 Process has finished.

Although started automatically, it is always possible to force a synchronization of the performance counters with the CIM repository content by using, for Windows 2000:

```
C:\>Winmgmt /resyncperf <PID_of_WINMGMT.EXE>
```

For Windows XP and Windows Server 2003, use the following:

```
C:\Winmgmt /resyncperf
```

In WMI, the *Win32_PerfFormattedData* is a calculated performance counter coming from the corresponding *Win32_PerfRawData* class representing the raw counter performance. The correspondence between the cooked class and the raw class is defined with the *AutoCook_RawClass* qualifier set on the same-named *Win32_PerfFormattedData* class. Table 3.74 shows the corresponding classes. Note that all *Win32_PerfFormattedData* classes have a corresponding *Win32_PerfRawData* class. For example, the *Win32_PerfFormattedData_Tcpip_NetworkInterface* has the *AutoCook_RawClass* qualifier set with a value equal to the name of the *Win32_PerfRawData_Tcpip_NetworkInterface* class.

Because the *Win32_PerfFormattedData_Tcpip_NetworkInterface* properties data is calculated from the *Win32_PerfRawData_Tcpip_NetworkInterface* properties data, this implies the use of a formula for each of the properties exposed by the *Win32_PerfFormattedData_Tcpip_NetworkInterface* class. The formula type is defined in a property qualifier for each calculated property of the *Win32_PerfFormattedData_Tcpip_NetworkInterface* class. This qualifier is called the *CookingType* qualifier and contains a cooking type identifier. Table 3.76 lists the *Win32_PerfFormattedData_Tcpip_NetworkInterface* properties exposing cooked counter information with the associated formula.

Table 3.77 lists the various cooking types available (for more details, you can refer to the *WinPerf.h* file, which comes with the Platform SDK installation).

The biggest challenge when working with the *Win32_PerfFormattedData* classes is not to extract information from the counters but to correctly interpret the meaning of the collected numbers. This is why it is important to explore in detail the miscellaneous *Win32_PerfFormattedData* derived classes with their properties. This information can be retrieved with the *LoadCIMInXL.wsf* script developed at the end of Chapter 4 of the first book, *Understanding WMI Scripting*, and available in the appendix. However, because a great number of details are provided in the qualifiers, it is best to run the script with a maximum of requested details. The command line to use would be as follows:

```
C:\>LoadCIMInXL.Wsf Win32_PerfFormattedData_Tcpip_NetworkInterface /Level:6
```

Table 3.76 The Win32_PerfFormattedData_Tcpip_NetworkInterface Properties

Name	Type	Cooking Type	Comments
BytesReceivedPersec	uint32	PERF_COUNTER_COUNTER	Bytes Received/sec is the rate at which bytes are received over each network adapter, including framing characters. Network Interface\Bytes Received/sec is a subset of Network Interface\Bytes Total/sec.
BytesSentPersec	uint32	PERF_COUNTER_COUNTER	Bytes Sent/sec is the rate at which bytes are sent over each network adapter, including framing characters. Network Interface\Bytes Sent/sec is a subset of Network Interface\Bytes Total/sec.
BytesTotalPersec	uint64	PERF_COUNTER_BULK_COUNT	Bytes Total/sec is the rate at which bytes are sent and received over each network adapter, including framing characters. Network Interface\Bytes Received/sec is a sum of Network Interface\Bytes Received/sec and Network Interface\Bytes Sent/sec.
CurrentBandwidth	uint32	PERF_COUNTER_RAWCOUNT	Current Bandwidth is an estimate of the current bandwidth of the network interface in bits per second (BPS). For interfaces that do not vary in bandwidth or for those where no accurate estimation can be made, this value is the nominal bandwidth.
OutputQueueLength	uint32	PERF_COUNTER_RAWCOUNT	Output Queue Length is the length of the output packet queue (in packets). If this is longer than two, there are delays and the bottleneck should be found and eliminated, if possible. Since the requests are queued by the Network Driver Interface Specification (NDIS) in this implementation, this will always be 0.
PacketsOutboundDiscarded	uint32	PERF_COUNTER_RAWCOUNT	Packets Outbound Discarded is the number of outbound packets that were chosen to be discarded even though no errors had been detected to prevent transmission. One possible reason for discarding packets could be to free up buffer space.
PacketsOutboundErrors	uint32	PERF_COUNTER_RAWCOUNT	Packets Outbound Errors is the number of outbound packets that could not be transmitted because of errors.
PacketsPersec	uint32	PERF_COUNTER_COUNTER	Packets/sec is the rate at which packets are sent and received on the network interface.
PacketsReceivedDiscarded	uint32	PERF_COUNTER_RAWCOUNT	Packets Received Discarded is the number of inbound packets that were chosen to be discarded even though no errors had been detected to prevent their delivery to a higher-layer protocol. One possible reason for discarding packets could be to free up buffer space.
PacketsReceivedErrors	uint32	PERF_COUNTER_RAWCOUNT	Packets Received Errors is the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.
PacketsReceivedNonUnicastPersec	uint32	PERF_COUNTER_COUNTER	Packets Received Non-Unicast/sec is the rate at which non-Unicast (subnet broadcast or subnet multicast) packets are delivered to a higher-layer protocol.
PacketsReceivedPersec	uint32	PERF_COUNTER_COUNTER	Packets Received/sec is the rate at which packets are received on the network interface.
PacketsReceivedUnicastPersec	uint32	PERF_COUNTER_COUNTER	Packets Received Unicast/sec is the rate at which (subnet) Unicast packets are delivered to a higher-layer protocol.
PacketsReceivedUnknown	uint32	PERF_COUNTER_RAWCOUNT	Packets Received Unknown is the number of packets received through the interface that were discarded because of an unknown or unsupported protocol.
PacketsSentNonUnicastPersec	uint32	PERF_COUNTER_COUNTER	Packets Sent Non-Unicast/sec is the rate at which packets are requested to be transmitted to non-Unicast (subnet broadcast or subnet multicast) addresses by higher-level protocols. The rate includes the packets that were discarded or not sent.
PacketsSentPersec	uint32	PERF_COUNTER_COUNTER	Packets Sent/sec is the rate at which packets are sent on the network interface.
PacketsSentUnicastPersec	uint32	PERF_COUNTER_COUNTER	Packets Sent Unicast/sec is the rate at which packets are requested to be transmitted to subnet-Unicast addresses by higher-level protocols. The rate includes the packets that were discarded or not sent.

Once loaded, all information related to the *Win32_PerfFormattedData_Tcpip_NetworkInterface* class with its properties and all associated qualifiers will be available in an Excel sheet.

Now that we have everything in our hands to understand what these raw and cooked counters are, the last step is to collect the counter information. Extracting a snapshot of a counter set is nice but not always relevant. The biggest interest in these classes is to monitor some counters over time and see

→ **Table 3.77** *The Miscellaneous Cooking Type*

Noncomputational Counter Types		
PERF_COUNTER_TEXT	2816	This counter type shows a variable-length text string in Unicode. It does not display calculated values.
PERF_COUNTER_RAWCOUNT	65536	Raw counter value requiring no further calculations and representing a single sample, which is the last observed value only.
PERF_COUNTER_LARGE_RAWCOUNT	65792	Same as PERF_COUNTER_RAWCOUNT but a 64-bit representation for larger values.
PERF_COUNTER_RAWCOUNT_HEX	0	Most recently observed value in hexadecimal format. It does not display an average.
PERF_COUNTER_LARGE_RAWCOUNT	256	Same as PERF_COUNTER_RAWCOUNT_HEX but a 64-bit representation in hexadecimal for use with large values.
Basic Algorithm Counter Types		
PERF_RAW_FRACTION	537003008	Ratio of a subset to its set as a percentage. This counter type displays the current percentage only, not an average over time.
PERF_SAMPLE_FRACTION	549585920	Average ratio of hits to all operations during the last two sample intervals. This counter type requires a base property with the PERF_SAMPLE_BASE counter type.
PERF_COUNTER_DELTA	4195328	This counter type shows the change in the measured attribute between the two most recent sample intervals.
PERF_COUNTER_LARGE_DELTA	4195584	Same as PERF_COUNTER_DELTA but a 64-bit representation for larger values.
PERF_ELAPSED_TIME	807666944	Total time between when the process started and the time when this value is calculated.
Counter Algorithm Counter Types		
PERF_AVERAGE_BULK	1073874176	Number of items processed, on average, during an operation. This counter type displays a ratio of the items processed (such as bytes sent) to the number of operations completed, and requires a base property with PERF_AVERAGE_BASE as the counter type.
PERF_COUNTER_COUNTER	272696320	Average number of operations completed during each second of the sample interval.
PERF_SAMPLE_COUNTER	4260864	Average number of operations completed in one second. This counter type requires a base property with the counter type PERF_SAMPLE_BASE.
PERF_COUNTER_BULK_COUNT	272696576	Average number of operations completed during each second of the sample interval. This counter type is the same as the PERF_COUNTER_COUNTER type, but it uses larger fields to accommodate larger values.
Timer Algorithm Counter Types		
PERF_COUNTER_TIMER	541132032	Average time that a component is active as a percentage of the total sample time.
PERF_COUNTER_TIMER_INV	557909248	Average percentage of time observed during sample interval that the object is not active. This counter type is the same as PERF_100NSEC_TIMER_INV except that it measures time in units of ticks of the system performance timer rather than in 100ns units.
PERF_AVERAGE_TIMER	805438464	Average time to complete a process or operation. This counter type displays a ratio of the total elapsed time of the sample interval to the number of processes or operations completed during that time. Requires a base property with PERF_AVERAGE_BASE as the counter type.
PERF_100NSEC_TIMER	542180608	Active time of one component as a percentage of the total elapsed time in units of 100ns of the sample interval.
PERF_100NSEC_TIMER_INV	592512256	Percentage of time the object was not in use. This counter type is the same as PERF_COUNTER_TIMER_INV except that it measures time in 100ns units rather than in system performance timer ticks.
PERF_COUNTER_MULTI_TIMER	574686464	Active time of one or more components as a percentage of the total time of the sample interval. Differs from PERF_100NSEC_MULTI_TIMER in that it measures time in units of ticks of the system performance timer, rather than in 100ns units. Requires a base property with the PERF_COUNTER_MULTI_BASE counter type.
PERF_COUNTER_MULTI_TIMER_INV	591463680	Inactive time of one or more components as a percentage of the total time of the sample interval. Differs from PERF_100NSEC_MULTI_TIMER_INV in that it measures time in units of ticks of the system performance timer, rather than in 100ns units. Requires a base property with the PERF_COUNTER_MULTI_BASE counter type.
PERF_100NSEC_MULTI_TIMER	575735040	This counter type shows the active time of one or more components as a percentage of the total time (100ns units) of the sample interval. Requires a base property with the PERF_COUNTER_MULTI_BASE counter type.
PERF_100NSEC_MULTI_TIMER_INV	592512256	Inactive time of one or more components as a percentage of the total time of the sample interval. Counters of this type measure time in 100ns units. Requires a base property with the PERF_COUNTER_MULTI_BASE counter type.
PERF_OBJ_TIME_TIMER		A 64-bit timer in object-specific units.
Precision Timer Algorithm Counter Types		
PERF_PRECISION_SYSTEM_TIMER	541525248	Similar to PERF_COUNTER_TIMER except that it uses a counter defined time base instead of the system timestamp.
PERF_PRECISION_100NS_TIMER	542573824	Similar to PERF_100NSEC_TIMER except that it uses a 100ns counter defined time base instead of the system 100ns timestamp.

Table 3.77 The Miscellaneous Cooking Type (continued)

Queue-length Algorithm Counter Types		
PERF_COUNTER_QUEUELEN_TYPE	4523008	Average length of a queue to a resource over time. It shows the difference between the queue lengths observed during the last two sample intervals divided by the duration of the interval.
PERF_COUNTER_LARGE_QUEUELEN_TYPE	4523264	Average length of a queue to a resource over time. Counters of this type display the difference between the queue lengths observed during the last two sample intervals, divided by the duration of the interval.
PERF_COUNTER_100NS_QUEUELEN_TYPE	5571840	Average length of a queue to a resource over time in 100 nanosecond units.
PERF_COUNTER_OBJECT_TIME_QUEUELEN_TYPE	6620416	Time an object is in a queue.
Base Counter Types		
PERF_AVERAGE_BASE	1073939458	Base value used in calculation of PERF_AVERAGE_TIMER and PERF_AVERAGE_BULK counter types.
PERF_COUNTER_MULTI_BASE	1107494144	Base value used in calculation of PERF_COUNTER_MULTI_TIMER, PERF_COUNTER_MULTI_TIMER_INV, PERF_100NSEC_MULTI_TIMER, and PERF_100NSEC_MULTI_TIMER_INV counter types.
PERF_LARGE_RAW_BASE	1073939715	Base value found in calculation of PERF_RAW_FRACTION. 64 bits.
PERF_RAW_BASE	1073939459	Base value used in calculation of the PERF_RAW_FRACTION counter type.
PERF_SAMPLE_BASE	1073939457	Base value used in calculation of the PERF_SAMPLE_COUNTER and PERF_SAMPLE_FRACTION counter types.

the evolution of the miscellaneous values. For example, we can save a set of counters for a particular performance class instance and repeat the operation at a regular time interval. To do this, we can reuse some of the WMI features shown in the previous samples.

Samples 3.77 and 3.78 show how to retrieve one or more performance counter instances and save the snapshot taken at a regular time interval in a .CSV file for later review and analysis. For example, you may wish to use Excel to generate graphics from the captured data.

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

Usage: WMICounterMonitor.wsf /CounterName:value /Instance:value [/interval:value] [/Raw[+|-]]
           [.Machine:value] [/User:value] [/Password:value]
```

Options:

```
CounterName : List of counter names to capture.
Instance    : List of instance corresponding to the counter name
interval   : Time interval between counter reads (in seconds).
Raw         : Determine if the raw performance counter must be taken instead of the formatted one.
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)
```

Examples:

```
WMICounterMonitor.wsf /CounterName:Tcpip_NetworkInterface
                      /Instance:"Compaq NC3121 Fast Ethernet NIC"
WMICounterMonitor.wsf /CounterName: Tcpip_NetworkInterface
                      /Instance:"Compaq NC3121 Fast Ethernet NIC" /Log+
WMICounterMonitor.wsf /CounterName:Spooler_PrintQueue
                      /Instance:"Lexmark 4039 Plus PS" /Interval:10
WMICounterMonitor.wsf /CounterName: Spooler_PrintQueue,Tcpip_NetworkInterface"
                      /Instance:"Lexmark 4039 Plus PS,Compaq NC3121 Fast Ethernet NIC"
                      /Interval:10
WMICounterMonitor.wsf /CounterName:PerfOS_Processor /Instance:"_Total" /Interval:10
```

The script usage is pretty simple. Basically, it requires two mandatory command-line parameters:

- The **/CounterName** switch: It represents the name (`<service_name>_<object_name>`) used to form the WMI performance class name. For example, with the `Win32_PerfFormattedData_Tcpip_NetworkInterface` class, the name given on the command line is “`Tcpip_NetworkInterface`.”
- The **/InstanceName** switch: It represents the instance name of the requested counter instance. All performance classes (raw and cooked) are using a unique Key property called `name` or are singleton classes. For example, with the `Win32_PerfFormattedData_Tcpip_NetworkInterface` class, the instance name given on the command line could be “`Compaq NC3121 Fast Ethernet NIC`.”

As a result, the complete command line would be as follows:

```
C:\>WMICounterMonitor.wsf /CounterName:"Tcpip_NetworkInterface"  
/Instance:"Compaq NC3121 Fast Ethernet NIC"
```

It is possible to combine several performance classes with their respective instances. In such a case, the command line to use would be as follows:

```
C:\>WMICounterMonitor.wsf /CounterName:"Tcpip_NetworkInterface,Tcpip_NBTCConnection,PerfNet_Browser"  
/Instance:"Compaq NC3121 Fast Ethernet NIC,_Total,@"
```

In such a case, the script will retrieve all properties of:

- An instance called “`Compaq NC3121 Fast Ethernet NIC`” for the `Win32_PerfFormattedData_Tcpip_NetworkInterface` class.
- An instance called “`_Total`” for the `Win32_PerfFormattedData_Tcpip_NBTCConnection` class.
- An instance of the singleton `Win32_PerfFormattedData_PerfNet_Browser` class. Its name is “`@`,” since the class is a singleton class

Let's see how Samples 3.77 and 3.78 work. The script is using an asynchronous event notification. Sample 3.77 contains the initialization process of the event notification. Sample 3.78 contains the event sink routine handling the event notifications.

Sample 3.77

Capturing performance counter values (raw or cooked) at regular time intervals (Part I)

```
1:<?xml version="1.0"?>  
.:  
8:<package>  
9:  <job>  
...:  
13:    <runtime>
```

```
...  
30:    </runtime>  
31:  
32:    <script language="VBScript" src="..\Functions\CreateTextFileFunction.vbs" />  
33:    <script language="VBScript" src="..\Functions\ConvertStringInArrayFunction.vbs" />  
34:    <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />  
35:    <script language="VBScript" src="..\Functions\PauseScript.vbs" />  
36:  
37:    <object progid="WbemScripting.SWbemRefresher" id="objWMIRefresher" />  
38:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>  
39:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />  
40:    <script language="VBScript">  
41:        <![CDATA[  
...  
45:        Const cComputerName = "LocalHost"  
46:        Const cWMINameSpace = "Root/cimv2"  
47:        Const cWMITimerClass = "__IntervalTimerInstruction"  
48:        Const cTimerID = "MyTimerEvent"  
49:        Const cWMITimerQuery = "Select * From __TimerEvent Where TimerID='MyTimerEvent'"  
50:        Const cWMIPerfFormattedClass = "Win32_PerfFormattedData"  
51:        Const cWMIPerfRawClass = "Win32_PerfRawData"  
...:  
119:        Set objWMISink = WScript.CreateObject ("WbemScripting.SWbemSink", "SINK_")  
120:  
121:        objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault  
122:        objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate  
123:  
124:        Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _  
125:                                                       strUserID, strPassword)  
...:  
128:        ' -----  
129:        ReDim objLogFileName (UBound(arrayCounterName))  
130:        For intIndice = 0 To UBound (arrayCounterName)  
131:            If boolRaw Then  
132:                strWMIPerfClass = cWMIPerfRawClass & "_" & _  
133:                               arrayCounterName(intIndice)  
134:            Else  
135:                strWMIPerfClass = cWMIPerfFormattedClass & "_" & _  
136:                               arrayCounterName(intIndice)  
137:            End If  
138:  
139:            Set objWMIClass = objWMIServices.Get (strWMIPerfClass)  
140:            If Err.Number Then ErrorHandler (Err)  
141:  
142:            boolSingleton = objWMIClass.Qualifiers_.Item("Singleton").Value  
143:            If Err.Number Then  
144:                Err.Clear  
145:                boolSingleton = False  
146:            End If  
147:  
148:            If boolSingleton Then  
149:                objWMIRefresher.Add objWMIServices, _  
150:                               strWMIPerfClass & "=@"  
151:            Else  
152:                objWMIRefresher.Add objWMIServices, _  
153:                               strWMIPerfClass & ".Name=" & _  
154:                               arrayInstanceName(intIndice) & "'"  
155:            End If  
...:  
158:        ' -----  
159:        strLogFileName = strWMIPerfClass & "_" & _  
160:                           Year(Date) & _  
161:                           Right ("0" & Month(Date), 2) & _
```

```
162:                     Right ("0" & Day(Date), 2) & "-" & _
163:                     Right ("0" & Hour(Time), 2) & _
164:                     Right ("0" & Minute(Time), 2) & _
165:                     Right ("0" & Second(Time), 2) & ".CSV"
166:
167:     Set objLogFileName(intIndice) = CreateTextFile (strLogFileName)
168:
169:     varTemp = objWMIClass.Path_.RelPath
170:     Set objWMIPropertySet = objWMIClass.Properties_
171:     For Each objWMIProperty In objWMIPropertySet
172:         Select Case objWMIProperty.Name
173:             Case "Caption"
174:             Case "Description"
175:             Case "Name"
176:             Case Else
177:                 varTemp = varTemp & "," & objWMIProperty.Name
178:         End Select
179:     Next
...:
184:     WriteToFile objLogFileName(intIndice), varTemp
185: Next
186:
187: objWMITimerresher.AutoReconnect = True
188: objWMITimerresher.Refresh
189:
190: ' -----
191: Set objWMIClass = objWMIServices.Get (cWMITimerClass)
192:
193: Set objWMITimerInstance = objWMIClass.SpawnInstance_
194: objWMITimerInstance.TimerID = cTimerID
195: objWMITimerInstance.IntervalBetweenEvents = intInterval * 1000
196: objWMITimerInstance.Put_ wbemChangeFlagCreateOrUpdate Or _
197:                 wbemFlagReturnWhenComplete
...:
200: WScript.Echo "Counter timer created."
201:
202: ' -----
203: objWMIServices.ExecNotificationQueryAsync objWMISink, cWMITimerQuery
204: If Err.Number Then ErrorHandler (Err)
205:
206: WScript.Echo "Waiting for events..."
207:
208: PauseScript "Click on 'Ok' to terminate the script ..."
209:
210: WScript.Echo vbCRLF & "Cancelling event subscription ..."
211: objWMISink.Cancel
212:
213: ' -----
214: objWMITimerInstance.Delete_
...:
217: WScript.Echo vbCRLF & "Counter timer deleted."
218:
219: For intIndice = 0 To UBound (arrayCounterName)
220:     CloseTextFile objLogFileName (intIndice)
221: Next
...:
226: WScript.Echo "Finished."
227:
...:
...:
...:
```

Once the command-line parameters are defined and parsed, the script executes the WMI connection (lines 121 through 125). Right after the WMI connection, the script enters a loop (lines 130 through 185) to create several items used by the script:

- Based on the `/Raw` command-line parameter value, the script builds the class name to be used (lines 131 through 137). If the `/Raw` switch is set to False, the class name will be a `Win32_PerfFormattedData` class. If the `/Raw` switch is set to True, the class name will be a `Win32_PerfRawData` class. The counter name given on the command line and the desired class name are combined to give the performance class name used to retrieve performance counter instance values.
- From the performance class name, the script verifies if the `Singleton` qualifier is set to determine how the performance counter instance will be retrieved (lines 139 through 146).
- If the class is a singleton class, the script uses the appropriate syntax to retrieve the unique instance from the given singleton class (lines 149 and 150). If the class is not a singleton class, it retrieves the performance counter instance with the name given with the `/InstanceName` switch (lines 152 through 154). It is important to note that the performance counter instances are not immediately stored in an `SWBemObject` instance. Instead, they are stored in `SWbemRefresher` object. This will give us the facility, later in the script, to refresh all instances stored in the `SWbemRefresher` object. This will improve the performance in retrieving the counter information, since a refresh operation is faster than an object instantiation. We have seen how the `SWbemRefresher` object can be used in Chapter 5 of *Understanding WMI Scripting*.
- Once the performance counter class is instantiated and saved in the `SWbemRefresher` object, the script prepares the .CSV file name (lines 159 through 167). Once the .CSV file is created, the file header is also created from the various properties available from the class (lines 169 through 184).

Each of these operations is repeated in a loop for every performance counter specified with the `/CounterName` switch (lines 130 through 185). Next, the script initializes the `SWbemRefresher` object and ensures that it automatically attempts to reconnect to a remote provider if the connection is broken (lines 187 and 188).

To gather performance counter data at regular time intervals, the script creates an `_IntervalTimerInstruction` instance. We already used a similar technique in previous samples (Samples 3.60 and 3.61, “PINGing a system at regular time intervals”). From line 191 through 200, the `_IntervalTimerInstruction` instance is created. Next, at line 203, the WQL event query is submitted to WMI to receive timer event notifications in the event sink (lines 229 through 274). The corresponding `SWbemSink` object is created at line 119. Next, as in any previous asynchronous event notification sample in this book, the script is paused (line 208). Once the script is resumed, the timer event notification is canceled (line 211), and the timer event instance is deleted from the CIM repository (lines 214 through 217). Before ending the script execution, every .CSV file created for each performance counter is closed (lines 219 through 221).

What happens in the event sink routine? The answer is shown in Sample 3.78.

Sample 3.78

*Capturing performance counter values (raw or cooked) at regular time intervals
(Part II)*

```
...:  
...:  
...:  
227:  
228: ' -----  
229: Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)  
...:  
238:     intCounter = intCounter + 1  
239:  
240:     Wscript.Echo  
241:     WScript.Echo FormatDateTime(Date, vbLongDate) & " at " &  
242:             FormatDateTime(Time, vbLongTime) &  
243:             " sample #" & intCounter & ". "  
244:  
245:     objWMIRefresher.Refresh  
246:  
247:     For intIndice = 1 To objWMIRefresher.Count  
248:         Set objWMIRefresherItem = objWMIRefresher.Item(intIndice * 2)  
249:         Set objWMIInstance = objWMIRefresherItem.Object  
250:  
251:         varTemp = FormatDateTime(Date, vbShortDate) & " " &  
252:                 FormatDateTime(Time, vbLongTime)  
253:         Set objWMIPropertySet = objWMIInstance.Properties_  
254:         For Each objWMIProperty In objWMIPropertySet  
255:             Select Case objWMIProperty.Name  
256:                 Case "Caption"  
257:                 Case "Description"  
258:                 Case "Name"  
259:                 Case Else  
260:                     varTemp = varTemp & "," & objWMIProperty.Value  
261:             End Select  
262:         Next
```

```
...:  
268:     WriteToFile objLogFileName(intIndice - 1), varTemp  
269:  
270:     WScript.Echo " WMICounter '" & arrayCounterName(intIndice - 1) & _  
271:             "' saved."  
272:     Next  
273:  
274: End Sub  
275:  
276: []>  
277: </script>  
278: </job>  
279:</package>
```

First, the sink routine increments a counter (line 238) used to display the number of performance counter samples already taken (lines 241 through 243). Next, the script refreshes the performance instances stored in the **SWbemRefresher** object (line 245). For each instance available in the **SWbemRefresher** object, the script executes a loop (lines 247 through 272) to save the performance counter instance data (lines 251 through 268). Each property (lines 254 through 262) of the performance counter is properly formatted into a .CSV format (line 260). Next, the counter data is appended to the corresponding .CSV file (line 268).

Note that the timer event is defaulted to ten seconds. The command-line definition exposes a switch, called **/Interval**, to redefine the timer event interval. Although a smaller value can be used, it is important to avoid too small intervals, because they will create huge .CSV files and generate a lot of network traffic if the monitored system is remote. In the same way as the performance data capture is executed during the timer event sink execution, if the timer interval is small, the refreshed instances data may not correspond to the instance state at the timer event triggering. WMI sends the event notifications to the consumer when they are available. However, it is likely that the consumer processes the event notification more slowly than the event notifications arrive. In such a case, waiting requests are queued. In such circumstances, it is likely that event notifications are not immediately processed when they are triggered. This is the reason why the script takes the time of the **_IntervalTimerInstruction** system class stored in the **TIME_CREATED** property exposed by the **_TimerEvent** system class (lines 252 and 253) and not the date and time of the event sink execution (lines 242 and 243).

3.8.2 Performance Monitoring provider

The *Performance Monitoring* provider is an instance and property provider that returns the same data seen in the System Monitor application. How-

ever, since it is not a *High-performance* provider, it lacks the speed and capabilities available through *High-performance* providers. (See Table 3.78.)

→ **Table 3.78** *The Performance Monitoring Providers 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
Performance Monitoring Provider																
PerfProv	Root/PerfMON	X						X	X			X	X	X	X	
PerfPropProv	Root/PerfMON		X		X			X				X	X	X	X	

The *Performance Monitoring* provider is not automatically registered in the Operating System. The main reason is that Microsoft does not encourage its use anymore and suggests the use of the *High-performance* providers instead. Therefore, if it is necessary to access system performance data using this provider, it must be registered first. (See Sample 3.79.)

→ **Sample 3.79** *The MOF file to register the Performance Monitoring provider*

```

1:  #pragma namespace("\\\\.\\Root")
2:
3:  instance of __Namespace
4:
5:    Name = "PerfMON";
6:
7:
8:  #pragma namespace("\\\\.\\Root\\PerfMON")
9:
10: instance of __Win32Provider as $PMPInst
11:
12:   Name = "PerfProv";
13:   ClsId = "{f00b4404-f8f1-11ce-a5b6-00aa00680c3f}";
14:
15:
16: instance of __InstanceProviderRegistration
17:
18:   Provider = "__Win32Provider.Name=\"PerfProv\"";
19:   SupportsPut = FALSE;
20:   SupportsGet = TRUE;
21:   SupportsDelete = FALSE;
22:   SupportsEnumeration = TRUE;
23:
24:
25: instance of __Win32Provider as $PMPPProp
26:
27:   Name = "PerfPropProv";
28:   Clsid = "{72967903-68EC-11d0-B729-00AA0062CBB7}";
29:
```

```

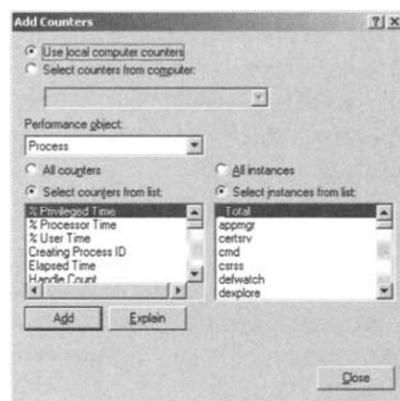
30:
31:     instance of __PropertyProviderRegistration
32:     {
33:         Provider = $PMPPProp;
34:         SupportsGet = TRUE;
35:         SupportsPut = FALSE;
36:     };
37:

```

Briefly, this MOF file registers the instance provider (lines 10 through 23) and the property provider (lines 25 through 36). This MOF file comes from the WMI section in the Microsoft Platform SDK. The only additions are lines 1 through 8 to ensure the registration of the provider in a dedicated namespace, in this case called PerfMON.

Once the provider is registered, it is also necessary to create the classes and the instances that will receive information from the provider. So, this provider is not usable out of the box and does not expose any classes by default. The classes must be created from a MOF file. For example, to retrieve all the Performance Monitor Counters related to the processes (as shown in Figure 3.54), the MOF file shown in Sample 3.80 must be loaded in the CIM repository.

Figure 3.54
The Performance Monitor Process counters.



Sample 3.80 A MOF file defining a class to retrieve Process counters from the Performance Monitor

```

1: #pragma namespace("\\\\.\\Root")
2:
3: instance of __Namespace
4: {
5:     Name = "PerfMON";
6: };
7:
8: #pragma namespace("//./Root/PerfMON")
9:

```

```
10: [dynamic, provider("PerfProv"), ClassContext("local|Process")]
11: class NTProcess
12: {
13:     [key]
14:     String Process;
15:
16:     [PropertyContext("% Privileged Time")]
17:         uint32 PercentagePrivilegedTime;
18:     [PropertyContext("% Processor Time")]
19:         uint32 PercentageProcessorTime;
20:     [PropertyContext("% User Time")]
21:         uint32 PercentageUserTime;
22:     [PropertyContext("Create Process ID")]
23:         uint32 CreateProcessID;
24:     [PropertyContext("Elapsed Time")]
25:         uint32 ElapsedTime;
26:     [PropertyContext("Handle Count")]
27:         uint32 HandleCount;
28:     [PropertyContext("ID Process")]
29:         uint32 ID;
30:     [PropertyContext("IO Data Bytes/sec")]
31:         uint32 IODataBytesPersec;
32:     [PropertyContext("IO Data Operations/sec")]
33:         uint32 IODataOperationsPersec;
34:     [PropertyContext("IO Other Bytes/sec")]
35:         uint32 IOOtherBytesPersec;
36:     [PropertyContext("IO Other Operations/sec")]
37:         uint32 IOOtherOperationsPersec;
38:     [PropertyContext("IO Read Bytes/sec")]
39:         uint32 IOReadBytesPersec;
40:     [PropertyContext("IO Read Operations/sec")]
41:         uint32 IOReadOperationsPersec;
42:     [PropertyContext("IO Write Bytes/sec")]
43:         uint32 IOWriteBytesPersec;
44:     [PropertyContext("IO Write Operations/sec")]
45:         uint32 IOWriteOperationsPersec;
46:     [PropertyContext("Page Faults/sec")]
47:         uint32 PageFaultsPersec;
48:     [PropertyContext("Page File Bytes")]
49:         uint32 PageFileBytes;
50:     [PropertyContext("Page File Bytes Peak")]
51:         uint32 PageFileBytesPeak;
52:     [PropertyContext("Pool Nonpaged Bytes")]
53:         uint32 PoolNonpagedBytes;
54:     [PropertyContext("Pool Paged Bytes")]
55:         uint32 PoolPagedBytes;
56:     [PropertyContext("Priority Base")]
57:         uint32 PriorityBase;
58:     [PropertyContext("Private Bytes")]
59:         uint32 PrivateBytes;
60:     [PropertyContext("Thread Count")]
61:         uint32 ThreadCount;
62:     [PropertyContext("Virtual Bytes")]
63:         uint32 VirtualBytes;
64:     [PropertyContext("Virtual Bytes Peak")]
65:         uint32 VirtualBytesPeak;
66:     [PropertyContext("Working Set")]
67:         uint32 WorkingSet;
68:     [PropertyContext("Working Set Peak")]
69:         uint32 WorkingSetPeak;
70: },
```

From line 1 through 8, the MOF file creates the `Root\PerfMON` namespace. It also creates the Performance Monitor `NTProcess` class (lines 10 through 69). The class and properties creations require three qualifiers specific to the *Performance Monitor* provider usage:

- *Provider* qualifier: This qualifier specifies the *Performance Monitoring* provider as the dynamic provider responsible for managing the data for the new class. This provider is specified at the class level (line 10).
- *ClassContext* qualifier: This qualifier specifies information needed by the *Performance Monitoring* provider to access the requested counter. The format of the *ClassContext* qualifier is:

```
machine | perfobject
```

where **machine** is the machine name and **perfobject** is the name of the performance object shown in System Monitor in Performance Monitoring (see Figure 3.55). In Sample 3.80, a value of “local | Process” indicates processes on the local machine in a class defined to hold counter data about the processes.

- *PropertyContext* qualifier: This qualifier identifies the display name of each counter. This is the name appearing in the Performance Monitoring tool (see Figure 3.53). This qualifier is specified for each property created for the class and accessing a specific Performance Monitor Counter (even lines 16 through 68).

Once compiled, **WMI CIM Studio** will show the screen seen in Figure 3.55.

Now, we are ready to access the Process counters from the Performance Monitor tool with a script. This last part has nothing unusual, since the script follows the traditional scripting techniques. It is available for your information in the Jscript Sample 3.81.

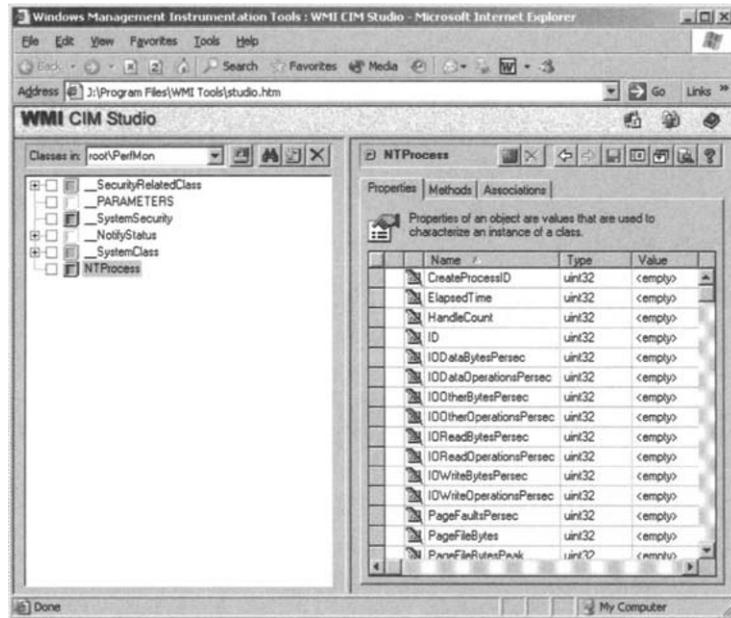
→ **Sample 3.81** *Viewing the Performance Monitor Process counters with a script*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
.:
13:   <script language="VBScript" src=".\\Functions\\DisplayFormattedPropertyFunction.vbs" />
14:
15:   <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:   <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
17:
18:   <script language="JScript">
19:     <![CDATA[
```

```

21:     var cComputerName = "LocalHost";
22:     var cWMINameSpace = "Root/PerfMON";
23:     var cWMIClass = "NTProcess";
24:
25:     ...
26:
27:     objWMIConnector.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault;
28:     objWMIConnector.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate;
29:     objWMIServices = objWMIConnector.ConnectServer(cComputerName, cWMINameSpace, "", "");
30:     objWMInstances = objWMIServices.InstancesOf (cWMIClass);
31:
32:
33:
34:     enumWMInstances = new Enumerator (objWMInstances);
35:     for (;! enumWMInstances.atEnd(); enumWMInstances.moveNext())
36:     {
37:         objWMInstance = enumWMInstances.item();
38:
39:         objWMIPropertySet = objWMInstance.Properties_
40:         enumWMIPropertySet = new Enumerator (objWMIPropertySet);
41:         for (;! enumWMIPropertySet.atEnd(); enumWMIPropertySet.moveNext())
42:         {
43:             objWMIProperty = enumWMIPropertySet.item()
44:
45:             DisplayFormattedProperty (objWMInstance,
46:                                     objWMIProperty.Name,
47:                                     objWMIProperty.Name,
48:                                     null);
49:
50:             WScript.Echo ();
51:         }
52:
53:     }]]>
54: </script>
55: </job>
56:</package>
```

Figure 3.55
The Process counters of the Performance Counters available from WMI.



3.9 Helper providers

Until now, each time we worked with a WMI provider, it collected information about a specific managed object in a computer. WMI also comes with a collection of providers designed to support some WMI internal features. In this section, we will examine the two most useful features from a WMI consumer perspective: the *View* provider to support the concept of views and the *Forwarding consumer* provider to perform events forwarding. We will also briefly examine the event Correlator providers.

3.9.1 The View provider

The main capability of the *View* provider is its ability to take properties from different source classes, different namespaces, and different computers and combine all information in one single class. Basically, the concept of the *View* provider looks very similar to the concept of views developed for the relational database. Of course, this comparison must be limited here, since its implementation is totally different. To take advantage of the *View* provider, it must first be registered in the CIM repository. The registration can be done with the MOFCOMP.EXE tool and the MOF file shown in Sample 3.82.

→ **Sample 3.82 Registering the View provider**

```
1:#pragma namespace("\\\\.\\Root")
2:
3:Instance of __Namespace
4:{ 
5:  Name = "View";
6:};
7:
8:#pragma namespace("\\\\.\\ROOT\\View")
9:
10:Instance of __Win32Provider as $DataProv
11:{ 
12:  Name = "MS_VIEW_INSTANCE_PROVIDER";
13:  ClsId = "{AA70DDF4-E11C-11D1-ABB0-00C04FD9159E}";
14:  ImpersonationLevel = 1;
15:  PerUserInitialization = "True";
16:  HostingModel = "NetworkServiceHost";
17:};
18:
19:Instance of __InstanceProviderRegistration
20:{ 
21:  Provider = $DataProv;
22:  SupportsPut = TRUE;
23:  SupportsGet = TRUE;
24:  SupportsDelete = TRUE;
25:  SupportsEnumeration = TRUE;
```

```

26: QuerySupportLevels = {"WQL:UnarySelect"};
27:};
28:
29:Instance of __MethodProviderRegistration
30:{ 
31: Provider = $DataProv;
32:};

```

You will note that the MOF file creates a new namespace for the provider (lines 3 through 6). This new namespace is not mandatory, since the registration can be done in any existing namespace. However, using a dedicated namespace clarifies the CIM repository organization, since only the classes created for the *View* provider will be available from this namespace. By default, there are few classes in the CIM repository supported by the *View* provider. These created classes are called the View classes. For example, the WMI information about the Internet Information Server (IIS) takes advantage of the *View* provider by accessing the *Win32_Service* instances in the *Root\CIMv2* namespace from the *Root\MicrosoftIISv2* namespace. The *View* provider is the component handling the cross-namespace access. As shown in Table 3.79, the *View* provider is implemented as an instance and a method provider.

Table 3.79*The View Providers 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
View Provider																
ViewProv	Not defined	X X						X X	X X	X X	X X					

As we will see, the *View* provider and the creation of the View classes are not directly related to the WMI scripting. However, once the View classes are created, any script can benefit from their existence.

We distinguish three types of view classes:

- The **Join view classes**: These classes represent the instances of different classes connected by a common property value.
- The **Union view classes**: These classes represent the union of one or more classes across the namespace boundary.
- The **Association view classes**: These classes represent views of existing association classes across the namespace boundary.

Because the View classes have these specific purposes, they use some specific qualifiers for their creation (see Table 3.80).

Table 3.80*The View Provider Qualifiers*

Qualifier	Comments
Direct	Data type: Boolean Used with view association properties to prevent association references from being mapped to a view reference.
HiddenDefault	Data type: Boolean Default value for a view class property based on a source class property with a different default value. The underlying source class is implied by the view.
JoinOn	Data type: string Defines how source class instances should be joined in join view classes. The following example shows how to use the JoinOn qualifier to join two source classes from the Performance Monitoring Provider.
MethodSource	Data type: string array Source method to execute for the view method. For similar syntax, see PropertySources Qualifier. The signature of the method must match the signature of the source class exactly. Copy the method signature from the MOF file that defines the source class. This qualifier is only valid when it is used with union views.
PostJoinFilter	Data type: string WQL query to filter instances after they have been joined in a join class.
PropertySources	Data type: string array Source properties from which a view class property gets data.
Union	Data type: Boolean Indicates whether you are defining a union class. Union views contain instances based on the union of source instances. For example, you might declare the following.
ViewSources	Data type: string array Set of WMI Query Language (WQL) queries that define the source instances and properties used in a specific view class. Positional correspondence of all the array qualifiers is important.
ViewSpaces	Data type: string array Namespaces where the source instances are located.

Sample 3.83 shows a MOF file creating a Join view class. In this sample, the *View* provider joins the *Win32_PerfRawData_PerfProc_Process* class with the *Win32_PerfRawData_PerfProc_Thread* class. Because the join requires a common property value between the two classes, the *IDProcess* property is used as the common property.

Sample 3.83*The Join View class*

```

1:#pragma namespace("\\\\.\\Root\\View")
2:
3:[
4: JoinOn("Win32_PerfRawData_PerfProc_Process.IDProcess=
           Win32_PerfRawData_PerfProc_Thread.IDProcess"),
5: ViewSources{
6:     "SELECT Name, IDProcess, PriorityBase
        FROM Win32_PerfRawData_PerfProc_Process" ,
7:     "SELECT Name, IDProcess, ThreadState, PriorityCurrent
        FROM Win32_PerfRawData_PerfProc_Thread"
8: },
9: ViewSpaces{
10:    "\\\\.\\"Root\\cimv2",
11:    "\\\\.\\"Root\\cimv2"
12: },
13: dynamic: ToInstance,
14: provider("MS_VIEW_INSTANCE_PROVIDER")
15: ]
16:
17:class JoinedProcessThread

```

```
18:{  
19: [read, PropertySources("IDProcess", "IDProcess")] UInt32 ProcessID;  
20: [read, PropertySources("Name", "")] String ProcessName;  
21: [read, PropertySources("", "Name"), key] String ThreadName;  
22: [read, PropertySources("", "ThreadState")] UInt32 State;  
23: [read, PropertySources("PriorityBase", "")] UInt32 BasePriority;  
24: [read, PropertySources("", "PriorityCurrent")] UInt32 CurrentPriority;  
25:};
```

At line 4, we see the *JoinOn* qualifier stating the link between the two classes on the *IDProcess* property. Because the MOF file links two instances coming from two different classes, lines 5 through 8 use the *ViewSources* qualifier to locate each instance coming from each respective class. This qualifier contains the WQL queries to locate instances of each class. Since every instance must be located in its respective namespaces, the *ViewSpaces* qualifier contains the WMI namespaces in which to look. At line 13, the View class is defined as a dynamic View class with the *Dynamic* qualifier. Next, the provider qualifier defines the *View* provider to support this new class definition.

From line 17 through 25, the class itself is defined. Each property of the class definition uses the *PropertySources* qualifier to map the desired property of the original class to a property of the View class. Note that in this example, each qualifier supported by the *View* provider is an array containing two items, where each of them corresponds to information related to the original classes. Once the MOF file is loaded in the CIM repository, the *JoinedProcessThread* class will be accessible from the *Root\View* namespace (see Figure 3.56).

We can reuse the script developed in Chapter 1 (Sample 1.5, “Listing all instances of a class with their properties formatted”) to display instances of this View class:

```
1: C:\>GetCollectionOfInstances.wsf JoinedProcessThread /NameSpace:Root\View  
2: Microsoft (R) Windows Script Host Version 5.6  
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
4:  
5:  
6: BasePriority: ..... 0  
7: CurrentPriority: ..... 0  
8: ProcessName: ..... Idle  
9: ProcessID: ..... 0  
10: State: ..... 2  
11: *ThreadName: ..... Idle/0  
12:  
13: BasePriority: ..... 0  
14: CurrentPriority: ..... 0  
15: ProcessName: ..... Idle  
16: ProcessID: ..... 0  
17: State: ..... 0  
18: *ThreadName: ..... _Total/_Total  
19:
```

```

20: BasePriority: ..... 8
21: CurrentPriority: ..... 0
22: ProcessName: ..... System
23: ProcessID: ..... 4
24: State: ..... 1
25: *ThreadName: ..... System/0
26:
27: BasePriority: ..... 8
28: CurrentPriority: ..... 13
29: ProcessName: ..... System
30: ProcessID: ..... 4
31: State: ..... 5
32: *ThreadName: ..... System/1
33:
34: BasePriority: ..... 8
35: CurrentPriority: ..... 13
36: ProcessName: ..... System
...
...
...

```

If you compare the properties of each original class with the content of the MOF file sample, you will see that each instance contains a mix of the properties coming from the *Win32_PerfRawData_PerfProc_Process* and *Win32_PerfRawData_PerfProc_Thread* classes joined by the *IDProcess* property. In other words, you consolidate the data from two different instances into one.

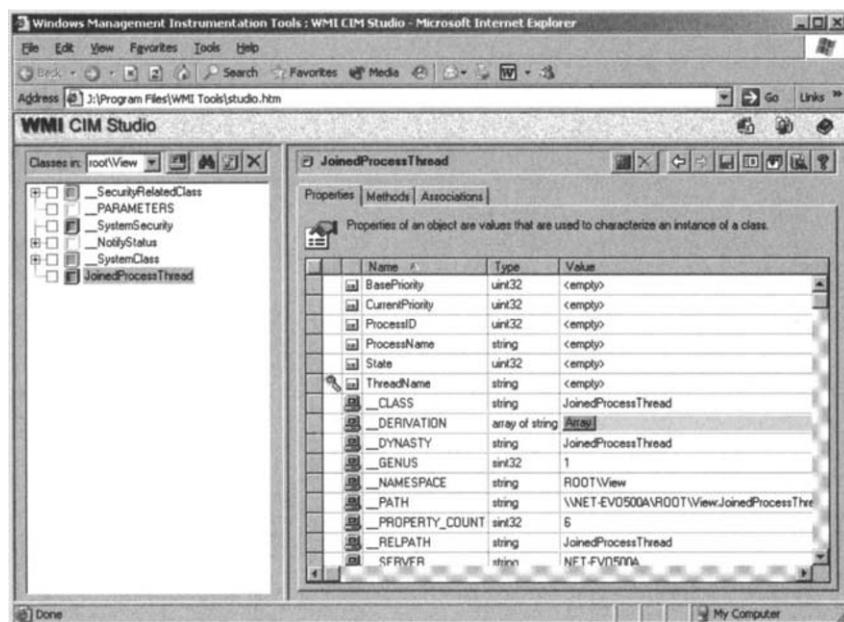


Figure 3.56 The new created Join View class.

The second View class type is the Union class. Sample 3.84 shows a MOF file example to create a Union View class.

→ **Sample 3.84** *The Union View class*

```
1:#pragma namespace ("\\\\\\\\Root\\\\View")
2:
3:[
4: Union,
5: ViewSources{
6:     "SELECT * From Win32_LogicalDisk Where DeviceID='C:'",
7:     "SELECT * From Win32_LogicalDisk Where DeviceID='C:'"
8: },
9: ViewSpaces{
10:    "\\\\net-dpen6400a.LissWare.Net\\\\Root\\\\CIMv2",
11:    "\\\\net-dpep6400.Emea.LissWare.Net\\\\Root\\\\CIMv2"
12: },
13: dynamic: ToInstance,
14: provider("MS_VIEW_INSTANCE_PROVIDER")
15:]
16:
17:Class UnionDrives_C
18:{  
19: [read, PropertySources("Description", "Description")] string Description;
20: [read, PropertySources("DeviceID", "DeviceID"), key] String DeviceID;
21: [read, PropertySources("VolumeSerialNumber", "VolumeSerialNumber"), key] string VolumeSN;
22: [read, PropertySources("FileSystem", "FileSystem")] String FileSystem;
23: [read, PropertySources("FreeSpace", "FreeSpace")] uint64 FreeSpace;
24: [read, PropertySources("VolumeName", "VolumeName"), key] String VolumeName;
25: [read, PropertySources("__SERVER", "__SERVER"), key] String Server;
26:};
```

The *Union* qualifier doesn't use any parameter. However, it also works in conjunction with the *ViewSources* and the *ViewSpaces* qualifiers (lines 5 and 9). Although the *ViewSources* qualifier continues to use a WQL statement to locate the required instances (lines 5 through 8), it is important to note the following:

- The *ViewSources* qualifier selects the “C:” drive instances only (see WQL data query).
- The *ViewSpaces* qualifier locates these instances in the same namespaces but on two different systems (lines 9 through 12).

As a result, the *UnionDrives_C* class (defined from line 17 through 25) will list a collection of two “C:” drive instances coming from the two different systems defined in the *ViewSpaces* qualifier. Note the presence of the *__SERVER* system property, exposed as the “server” Key property (line 25), and the definition of the *VolumeSerialNumber* property as another Key property to distinguish each View class instance (line 24). Originally, the Key property of the *Win32_LogicalDisk* source class was the *DeviceID* property, which is the drive letter of the logical disk. Now, because the View class

retrieves a collection of “C:” drive instances, the *DeviceID* property can’t be used, since it is the only Key property originally used to distinguish the various instances. This means that other uniqueness criteria must be created. This is why the *server* and the *VolumeSerialNumber* properties are also defined as Key properties. The output would be as follows:

```

1:  C:\>GetCollectionOfInstances.wsf UnionDrives_C /NameSpace:Root\View
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
6:  Description: ..... Local Fixed Disk
7:  *DeviceID: ..... C:
8:  FileSystem: ..... NTFS
9:  FreeSpace: ..... 827183616
10: Server: ..... NET-DOPEN6400A
11: *VolumeName: ..... Windows 2003
12: *VolumeSerialNumber: ..... 988BD271
13:
14: Description: ..... Local Fixed Disk
15: *DeviceID: ..... C:
16: FileSystem: ..... NTFS
17: FreeSpace: ..... 725999616
18: Server: ..... NET-DPEP6400
19: *VolumeName: ..... Windows 2003
20: *VolumeSerialNumber: ..... 74052931

```

By addressing one single class from one namespace, we get a collection of instances coming from two different systems. However, it looks like this information is coming from the local system. It must be clear that the credentials used to connect to the **Root\View** namespace must also be valid to connect to the remote **Root\CIMv2** namespaces!

The last View class type supported by the *View* provider is the Association view class. Basically, what we did before with the previous classes (see Sample 3.84) can be done with the association classes. This class creation is illustrated in Sample 3.85.

Sample 3.85 The Association View class

```

1:#pragma namespace ("\\\\.\\\\Root\\\\View")
2:
3:[
4: Union,
5: ViewSources{
6:         "SELECT * From Win32_LogicalDisk Where DeviceID='C:'"
7:         },
8: ViewSpaces{
9:         "\\\\.\\\\Root\\\\CIMv2"
10:        },
11: dynamic,
12: provider("MS_VIEW_INSTANCE_PROVIDER")

```

```
13: ]
14:
15:Class UnionDrive_C
16:{  
17: [read, PropertySources("Description")] string Description;
18: [read, PropertySources("DeviceID"), key] String DeviceID;
19: [read, PropertySources("VolumeSerialNumber"), key] string VolumeSerialNumber;
20: [read, PropertySources("FileSystem")] String FileSystem;
21: [read, PropertySources("FreeSpace")] uint64 FreeSpace;
22: [read, PropertySources("VolumeName"), key] String VolumeName;
23: [read, PropertySources("__SERVER"), Key] String Server;
24:
25: [Implemented,
26: MappingStrings("Fmifs.dll | Method ChkDskExRoutine"): ToSubClass,
27: MethodSource("Chkdsk")] uint32 ViewChkdsk([in] boolean FixErrors = FALSE,
28:                                         [in] boolean VigorousIndexCheck = TRUE,
29:                                         [in] boolean SkipFolderCycle = TRUE,
30:                                         [in] boolean ForceDismount = FALSE,
31:                                         [in] boolean RecoverBadSectors = FALSE,
32:                                         [in] boolean OkToRunAtBootUp = FALSE);
33:];
34:
35:[
36: Association,
37: ViewSources {
38:     "SELECT * FROM Win32_DiskQuota"
39: },
40: ViewSpaces {
41:     "\\\\.\\"Root\\\\CIMv2"
42: },
43: dynamic,
44: provider("MS_VIEW_INSTANCE_PROVIDER")
45:]
46:
47:class Association_UnionOfDriveQuota
48:{  
49:     [key, PropertySources("QuotaVolume")]
50:     UnionDrive_C ref Drive;
51:
52:     [Direct, key, PropertySources("User")]
53:     Win32_Account ref User;
54:};
```

The MOF file sample contains two main parts:

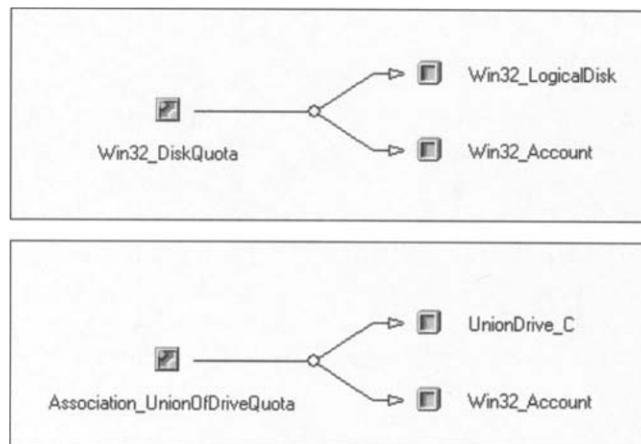
- A Union View class creation, made from the *Win32_LogicalDisk* class (lines 3 through 33). Basically, this Union View class creation follows the same rules as Sample 3.84. However, the instances of this class are coming from one single namespace in one single computer (lines 8 through 10). So, this technique maps a class from one namespace (**Root\CIMv2**) to make it available to another namespace (**Root\View**). The Association View class created in the second part of the MOF file references this Union class.

The interesting addition in this Union class is the mapping of the *Chkdsk* method defined in the *Win32_LogicalDisk* class. In the previous sample, we only mapped some properties. In this example, we also map a method with the help of the *MethodSource* qualifier. Note that it is important to use a method definition in the View Class that exactly matches the method definition in the original class, which is, in this example, the *Win32_LogicalDisk* class. The best method is to take a MOF file export of this class, cut the method definition from this exported MOF file, and paste it into your MOF file. As a result, the *Chkdsk* method is mapped to the View class as the *ViewChkdsk* method (lines 27 through 32).

- An Association View class, made from the *Win32_DiskQuota* association class, associates the previously created *UnionDrive_C* View class with the *Win32_Account* class. The link with the *Win32_Account* class is defined with the *User* reference originally coming from the *Win32_DiskQuota* class (lines 35 through 54). If you go back to the *Win32_DiskQuota* association class and look at what this class associates, you will see that two classes are linked together:
 - The *Win32_LogicalDisk* class with the *QuotaVolume* reference
 - The *Win32_Account* class with the *User* reference

However, the *QuotaVolume* reference is overwritten by the *Drive* reference defined in the association View class, while the *User* reference is kept intact in the *Win32_DiskQuota* association class by the presence of the *Direct* qualifier. You can compare the *Association_UnionOfDriveQuota* association View class with the *Win32_DiskQuota* association class shown in Figure 3.57.

Figure 3.57
The *Win32_DiskQuota* association class and the created Association View class.



From a scripting perspective, we obtain the following output for the newly created Association View class:

```

1: C:\>GetCollectionOfInstances.wsf Association_UnionOfDriveQuota /NameSpace:Root\View
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
6: *Drive: ..... \\NET-DOPEN6400A\Root\VIEW:UnionDrive_C.DeviceID=
   "C:",Server="NET-DOPEN6400A",VolumeName="Windows 2003",VolumeSerialNumber="988BD271"
7: *User: ..... Win32_Account.Domain=
   "NET-DOPEN6400A",Name="Administrators"

```

This last example illustrates how it is possible to associate classes from different namespaces with the *View* provider.

3.9.2 The Forwarding consumer provider

The *Microsoft WMI Forwarding consumer* provider is available under Windows XP.

Registered in the **Root\Subscription** namespace, it allows local WMI events to be forwarded to remote systems. In remote systems the *Microsoft WMI Forwarding event* provider, registered in the **Root\CIMv2** namespace, triggers the received forwarded event as an instance of the *MSFT_ForwardedEvent* extrinsic event class. To trace the *Microsoft WMI Forwarding consumer* provider activity, the *Microsoft WMI Forwarding Consumer Trace* event provider, also registered in the **Root\Subscription** namespace, generates WMI events with a specific set of extrinsic event classes to any WMI consumers subscribing to these event notifications.

Table 3.81 The WMI Forwarding Consumer and Forwarding Event Providers Capabilities

Provider Name	Provider Name Space	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
Microsoft WMI Forwarding Consumer Provider						X						X				
FwdProv	Root/Subscription												X			
Microsoft WMI Forwarding Consumer Trace Event Provider						X							X			
FwdProv	Root/Subscription												X			
Microsoft WMI Forwarding Event Provider														X		
FwdProv	Root/CIMV2						X							X		

Table 3.81 summarizes the characteristics of the *Microsoft WMI Forwarding Consumer*, *Microsoft WMI Forwarding Event*, and *Microsoft WMI*

Forwarding Consumer Trace event providers. Table 3.82 lists classes supported by these three providers implemented in one single DLL (*FwdProv.DLL*).

Table 3.82

The WMI Forwarding Consumer and Forwarding Event Providers Classes

Microsoft WMI Forwarding Consumer Provider class	
Name	Description
MSFT_ForwardingConsumer	Represents an event consumer that forwards messages to target computer(s).
Microsoft WMI Forwarding Consumer Trace Event Provider classes	
Name	Description
Select * From MSFT_FCTraceEventBase	
MSFT_FCTraceEventBase	Base class for all forwarding consumer trace events.
MSFT_FCExecutedTraceEvent	Represents an execution of the forwarding consumer.
MSFT_FCTargetTraceEvent	Represents an attempt to forward a message to a target.
Microsoft WMI Forwarding Event Provider class	
Name	Description
MSFT_ForwardedMessageEvent	Base class for all forwarded messages.
MSFT_ForwardedEvent	Indicates the arrival of a forwarded event.

Event forwarding involves at the minimum two systems: a source machine and a target machine. Figure 3.58 illustrates the logical organization of the various elements involved in event forwarding.

The source machine is the machine where the original WMI event notification occurs (i.e., *SourceComputer01*). This is also the machine that forwards the event. To do so, it is necessary to create an instance of the *MSFT_ForwardingConsumer* class supported by the *Microsoft WMI Forwarding consumer* provider. Basically, this instance determines which event notification will be forwarded and to which target machines it must be forwarded. Therefore, the *Microsoft WMI Forwarding consumer* provider consumes a specific WMI event notification to be forwarded to the target machine, as defined in the *MSFT_ForwardingConsumer* instance. At the other end, the target machine (i.e., *TargetComputer01*) is the system receiving the event forwarded by the *Microsoft WMI Forwarding consumer* provider in the source machine. The *Microsoft WMI Forwarding event* provider is in charge of receiving the forwarded event and triggering an event notification as an instance of the *MSFT_ForwardedEvent* extrinsic event class. Any WMI consumer who subscribed to receive the forwarded event will get a notification. More than simply notifying that an event has been forwarded, the subscriber will also find information inside the *MSFT_ForwardedEvent* about the instances (located in the source machine) subject to the event notification.

To configure this mechanism, the first step is to create an instance of the *MSFT_ForwardingConsumer* class with its *_EventFilter* associated class.

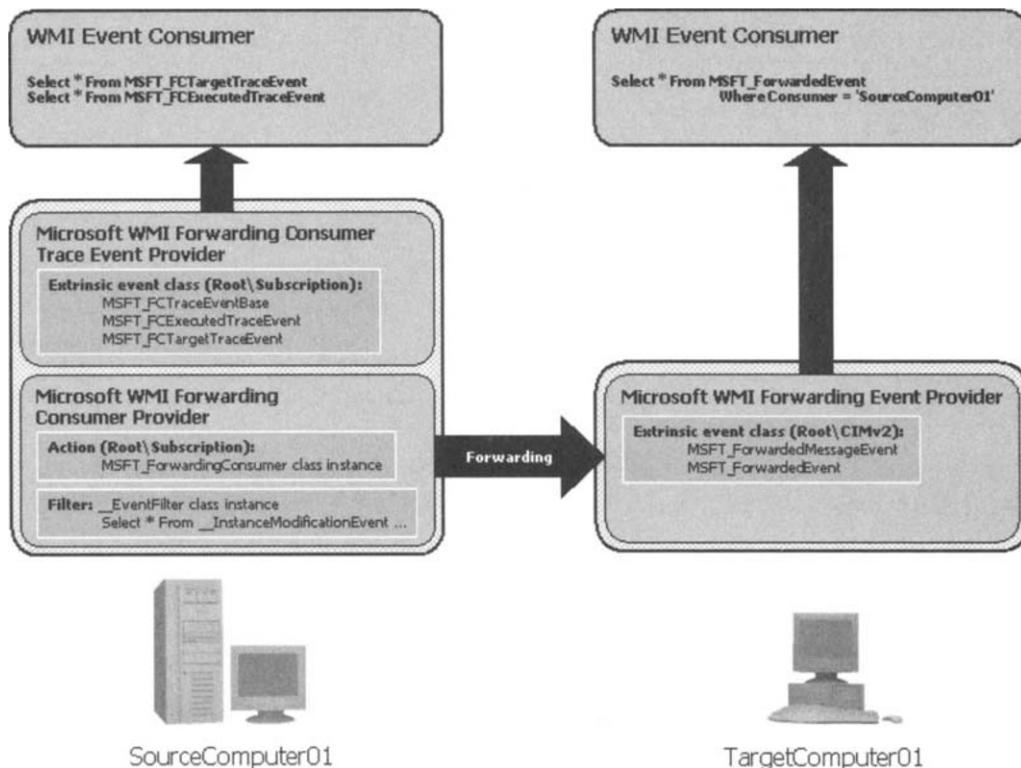


Figure 3.58 The WMI Forwarding providers roles and locations.

This can be done with a MOF file (see Sample 3.86). Of course, a script can create these instances as well.

Sample 3.86 A MOF file to forward WMI events

```

1:#pragma namespace ("\\.\Root\\Subscription")
2:
3:Instance of MSFT_ForwardingConsumer as $SRCComputer01Consumer
4:
5:{
6:    // Indicates whether to include authentication information when forwarding the message.
7:    Authenticate = "True";
8:
9:    // Indicates whether to encrypt the message before forwarding.
10:   Encryption = "False";
11:
12:   // The Quality-of-Service used to forward the message (0=Synchronous).
13:   ForwardingQoS = 0;
14:
15:   // Indicates whether to send schema information for the event to be forwarded.
16:   IncludeSchema = "False";

```

```

17:
18:    // Name of the computer from which the message originates (inherited property).
19:    // MachineName =
20:
21:    // Maximum size of the queue in bytes.
22:    // MaximumQueueSize =
23:
24:    // A string uniquely identifying this consumer.
25:    Name = "SourceComputer01";
26:
27:    // An array of addresses to forward the messages to.
28:    // The forwarding consumer will try to forward the message, in order,
29:    // to each address in the list until it successfully sends it to one of them.
30:    Targets = {"NET-DPEN6400A.LissWare.NET"};
31:
32:    // A security descriptor, in SDDL format, that is attached to the
33:    // forwarded event when it is raised on the receiving end.
34:    // This security descriptor indicates to the receiver which security
35:    // identitifiers are allowed to consume the forwarded event.
36:    TargetSD = "";
37:};
38:
39:Instance of __EventFilter as $SRCComputer01Filter
40:
41:{ 
42:    Name = "Forward all Win32_Service instance modifications";
43:    Query = "Select * From __InstanceModificationEvent Within 5 "
44:            "Where TargetInstance ISA 'Win32_Service'";
45:    QueryLanguage = "WQL";
46:    EventNamespace = "Root\\CIMv2";
47:};
48:
49:Instance of __FilterToConsumerBinding
50:
51:{ 
52:    Consumer=$SRCComputer01Consumer;
53:    Filter=$SRCComputer01Filter;
54:};

```

As with any WMI consumer providers, Sample 3.86 creates:

- An instance of the *MSFT_ForwardingConsumer* class (derived from the *__EventConsumer* superclass) supported by the WMI consumer (lines 3 through 37)
- An instance of the *__EventFilter* system class to define the WMI event notification to subscribe to (lines 39 through 47)
- An instance of the *__FilterToConsumerBinding* association class to link the consumer with the WQL event notification filter (lines 49 through 54)

In this particular example, we are interested in the *MSFT_ForwardingConsumer* instance creation, since it contains all parameters related to the forwarding mechanism.

- **Authenticate** (line 7): **Authenticate** indicates whether to include authentication information when forwarding the message. Is there a reason not to include the authentication information? Well, in some situations, it could be necessary to forward events between Active Directory Forests, where no trust exists between Forests. In such a case, it is necessary to set the **Authenticate** parameter to False. However, this is not enough to get this working. Since the authentication information is not contained in the message, to avoid an access-denied error message and ensure that the *Microsoft WMI Forwarding Event* in the target machine effectively triggers the event notification, it is necessary to set the *AllowUnauthenticatedEvents* registry key located in the HKLM\SOFTWARE\Microsoft\WBEM\FWD hive to 1. This will ensure that the *Microsoft WMI Forwarding event* provider accepts unauthenticated events. Note that if you change this registry key value, it is necessary to restart the **WinMgmt.Exe** service to make the change effective.
- **Encryption** (line 10): **Encryption** indicates that packets between the source and the target machine are encrypted. The encryption is based on the COM Packet Privacy encryption mechanism. Note that by default, under Windows XP, the encryption is set to False.
- **ForwardingQoS** (line 13): This property is not used, but it is reserved for future use and extensions. Its default value must be zero.
- **IncludeSchema** (line 16): The **IncludeSchema** purpose is to send a CIM repository definition (which exists in the source machine) of the instances subject to the event. This ensures that the target machine understands the structure of the *TargetInstance* and *PreviousInstance* properties containing objects returned by the event forwarding notification in case these class definitions do not exist in the target machine. Keep in mind that setting **IncludeSchema** to True will represent a big overhead for the network. Therefore, it is best to set **IncludeSchema** to False for performance reasons. Note that setting **IncludeSchema** to False always includes the schema information for the first event with the condition that events are from the same source and class.
- **MaximumQueueSize** (line 22): This property is reserved for future use and extensions. It should not be defined (the line is commented out in Sample 3.86).

- **Targets** (line 30): Targets contain the list of computers that must be contacted as targets. This property acts as a “try list,” which means that the forwarding consumer will try to forward the message, in order, to each address in the list until it successfully sends it to one of them. If you want to forward the same event to several computers, you must register several instances of the Forwarding Consumer (one per target). On the other hand, it is important to note that target machines, consuming forwarded events, are able to process *MSFT_ForwardedEvent* events from multiple computers and networks.
- **TargetSD** (line 36): TargetSD represents a security descriptor in security descriptor definition language (SDDL) format (see http://msdn.microsoft.com/library/en-us/security/Security/security_descriptor_string_format.asp for more information about SDDL). This security descriptor is attached to the forwarded event when it is raised on the target machine. It indicates to the receiver which security identifiers are allowed to consume the forwarded event.

Once the MOF file in Sample 3.86 is loaded with MOFCOMP.EXE in the CIM repository of the source machine, any *Win32_Service* instance modification events will be forwarded to the target machine. The only thing to do on the target machine is to run a WMI consumer subscribing to the event forwarding notification. In this example, we can reuse the **GenericEventAsyncConsumer.wsf** script (see Sample 6.17, “A generic script for asynchronous event notification” in the appendix). The WQL event query to formulate on the target machine must be as follows:

```
1: C:\>GenericEventAsyncConsumer.wsf "Select * From MSFT_ForwardedEvent
   Where Consumer = 'SourceComputer01'"
2: Microsoft (R) Windows Script Host Version 5.6
3: Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5: Waiting for events...
6:
7: BEGIN - OnObjectReady.
8: Thursday, 11 July, 2002 at 15:12:10: 'MSFT_ForwardedEvent' has been triggered.
9:     Account (wbemCimtypeUInt8) = 1
10:    Account (wbemCimtypeUInt8) = 5
11:    Account (wbemCimtypeUInt8) = 0
...:
34:    Account (wbemCimtypeUInt8) = 3
35:    Account (wbemCimtypeUInt8) = 0
36:    Account (wbemCimtypeUInt8) = 0
37:    Authenticated (wbemCimtypeBoolean) = True
38:    Consumer (wbemCimtypeString) = SourceComputer01
39:    Event (wbemCimtypeObject)
40:        PreviousInstance (wbemCimtypeObject)
41:            AcceptPause (wbemCimtypeBoolean) = False
42:            AcceptStop (wbemCimtypeBoolean) = False
43:            Caption (wbemCimtypeString) = SNMP Service
```

```

44:     CheckPoint (wbemCimtypeUInt32) = 0
45:     CreationClassName (wbemCimtypeString) = Win32_Service
46:     Description (wbemCimtypeString) = Enables Simple Network Management ...
...
60:     State (wbemCimtypeString) = Stopped
61:     Status (wbemCimtypeString) = OK
62:     SystemCreationClassName (wbemCimtypeString) = Win32_ComputerSystem
63:     SystemName (wbemCimtypeString) = NET-DPEP6400A
64:     TagId (wbemCimtypeUInt32) = 0
65:     WaitHint (wbemCimtypeUInt32) = 0
66:     SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
67:     TargetInstance (wbemCimtypeObject)
68:         AcceptPause (wbemCimtypeBoolean) = False
69:         AcceptStop (wbemCimtypeBoolean) = True
70:         Caption (wbemCimtypeString) = SNMP Service
71:         CheckPoint (wbemCimtypeUInt32) = 0
72:         CreationClassName (wbemCimtypeString) = Win32_Service
73:         Description (wbemCimtypeString) = Enables Simple Network Management ...
...
87:     State (wbemCimtypeString) = Running
88:     Status (wbemCimtypeString) = OK
89:     SystemCreationClassName (wbemCimtypeString) = Win32_ComputerSystem
90:     SystemName (wbemCimtypeString) = NET-DPEP6400A
91:     TagId (wbemCimtypeUInt32) = 0
92:     WaitHint (wbemCimtypeUInt32) = 0
93:     TIME_CREATED (wbemCimtypeUInt64) = (null)
94:     Machine (wbemCimtypeString) = NET-DPEP6400A
95:     Namespace (wbemCimtypeString) = ROOT\subscription
96:     SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
97:     Time (wbemCimtypeDatetime) = 11-07-2002 15:12:10 (20020711131210.000000+000)
98:     TIME_CREATED (wbemCimtypeUInt64) = 11-07-2002 13:12:10 (20020711131210.953590+120)
99:
100:    END - OnObjectReady.

```

There are two levels of information received:

- The information about the forwarded event itself (lines 9 through 38 and lines 94 through 98)
- The information about the instances subject to the event triggered in the source machine (lines 39 through 93), which is divided into two parts: the *PreviousInstance* (lines 40 through 65) and *TargetInstance* (lines 67 through 92)

In the source machine, with the help of the *Microsoft WMI Forwarding Consumer Trace* event provider and its set of supported classes, it is possible to trace the execution of the *Microsoft WMI Forwarding consumer* provider. By using a simple WMI consumer subscribing to the *MSFT_FCExecutedTraceEvent* or the *MSFT_FCTargetTraceEvent* extrinsic event classes, it is possible to gather information about an event forwarding execution. We can reuse the *GenericEventAsyncConsumer.wsf* script shown in Sample 6.17 (“A generic script for asynchronous event notification”) in the appendix to catch this event in the source machine:

```

1:  C:\>GenericEventAsyncConsumer.wsf "Select * From MSFT_FCTargetTraceEvent"
   /Namespace:Root\Subscription
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  Waiting for events...
6:
7:  BEGIN - OnObjectReady.
8:  Friday, 12 July, 2002 at 11:35:26: 'MSFT_FCTargetTraceEvent' has been triggered.
9:    Consumer (wbemCimtypeObject)
10:      Authenticate (wbemCimtypeBoolean) = True
11:      CreatorSID (wbemCimtypeUInt8) = 1
12:      CreatorSID (wbemCimtypeUInt8) = 2
13:      CreatorSID (wbemCimtypeUInt8) = 0
14:      CreatorSID (wbemCimtypeUInt8) = 0
15:      CreatorSID (wbemCimtypeUInt8) = 0
16:      CreatorSID (wbemCimtypeUInt8) = 0
17:      CreatorSID (wbemCimtypeUInt8) = 0
18:      CreatorSID (wbemCimtypeUInt8) = 5
19:      CreatorSID (wbemCimtypeUInt8) = 32
20:      CreatorSID (wbemCimtypeUInt8) = 0
21:      CreatorSID (wbemCimtypeUInt8) = 0
22:      CreatorSID (wbemCimtypeUInt8) = 0
23:      CreatorSID (wbemCimtypeUInt8) = 32
24:      CreatorSID (wbemCimtypeUInt8) = 2
25:      CreatorSID (wbemCimtypeUInt8) = 0
26:      CreatorSID (wbemCimtypeUInt8) = 0
27:      Encryption (wbemCimtypeBoolean) = False
28:      ForwardingQoS (wbemCimtypeSint32) = 0
29:      IncludeSchema (wbemCimtypeBoolean) = False
30:      MachineName (wbemCimtypeString) = (null)
31:      MaximumQueueSize (wbemCimtypeUInt32) = 65535
32:      *Name (wbemCimtypeString) = SourceComputer01
33:      Targets (wbemCimtypeString) = NET-DPEN6400A.LissWare.NET
34:      TargetSD (wbemCimtypeString) = (null)
35:      ExecutionId (wbemCimtypeString) = {F7EF6E36-FBAD-4C80-8196-BCC3B9A3A895}
36:      SECURITY_DESCRIPTOR (wbemCimtypeUInt8) = (null)
37:      StatusCode (wbemCimtypeUInt32) = 0
38:      Target (wbemCimtypeString) = 7879E40D-9FB5-450a-
   8A6D-00C89F349FC@ncacn_ip_tcp:NET-DPEN6400A.LissWare.NET
39:      TIME_CREATED (wbemCimtypeUInt64) = 12-07-2002 09:35:26 (20020712093526.907875+120)
40:  END - OnObjectReady.

```

The *MSFT_FCTargetTraceEvent* extrinsic event class represents an attempt to forward a message to a target machine, while the *MSFT_FCEExecutedTraceEvent* extrinsic event class represents an execution of the forward-

Table 3.83 The *StatusCode* Property Returned Values

	Value	Description
WMIMSG_E_AUTHFAILURE	0x80042101	There was a problem in attaching authentication info with the forwarded event.
WMIMSG_E_ENCRYPTFAILURE	0x80042102	There was a problem in encrypting the forwarded event.
WMIMSG_E_INVALIDADDRESS	0x80042105	The target address specified for the forwarded event is invalid.
WMIMSG_E_TARGETNOTFOUND	0x80042106	The machine for the specified target address was not found.
WMIMSG_E_INVALIDMESSAGE	0x80042108	A receiver has received an invalid / corrupt message.
WMIMSG_E_REQSVNOTAVAIL	0x80042109	A requested service is not available.
WMIMSG_E_TARGETNOTLISTENING	0x80042113	The target is valid but it is not listening for forwarded events.

ing consumer. Therefore, the latter contains information about instances related to the WQL event filter associated with the *MSFT_ForwardingConsumer* instance. In both cases, the most important property is the *Statuscode* property (line 37), since this value contains a return code determining if the event forwarding was successful (see Table 3.83).

3.9.3 The Event Correlator providers

The *Event Correlator* providers are available under Windows XP. The goal of this provider set is to correlate different events and send an alert or provide information only if a sequence of WMI events or a combination of an expected WMI data set is available from the system. For instance, monitoring a system and sending an alert only when the disk usage is at 50 percent and when the CPU usage has been higher than 80 percent in the last ten minutes for a period of two minutes is an example of correlation.

Although available in Windows XP, it is very important to note that today Microsoft does not encourage the use of these WMI correlation providers. This is the reason why Microsoft doesn't plan to make this provider available under Windows Server 2003 (although it was available during the beta program of Windows Server 2003).

Table 3.84 The Correlation WMI Providers

Provider Name	Provider Name Space	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
The Updating consumers																
Microsoft WMI Updating Consumer Assoc Provider	Root/subscription	X					X	X	X	X		X				
MSFT_UCScenarioAssociation																
Microsoft WMI Updating Consumer Event Provider	Root/subscription					X						X				
Select * From MSFT_UCTraceEventBase																
Select * From MSFT_UCEventBase																
Microsoft WMI Updating Consumer Provider	Root/subscription						X					X				
MSFT_UpdatingConsumer																
MSFT_UCScenarioControl																
The Template providers																
Microsoft WMI Template Association Provider	Root/subscription	X				X	X					X				
Microsoft WMI Template Event Provider	Root/subscription		X									X				
Select * From InstanceOperationEvent WHERE TargetInstance isa "MSFT_TemplateBase"																
Microsoft WMI Template Provider	Root/subscription	X				X	X	X	X			X				
The Transient providers																
Microsoft WMI Transient Event Provider	Root/subscription		X									X				
Select * From MSFT_TransientEggTimerEvent																
Select * From InstanceOperationEvent where TargetInstance isa "MSFT_TransientStateBase"																
Microsoft WMI Transient Provider	Root/subscription	X				X	X	X	X			X				
Microsoft WMI Transient Reboot Event Provider	Root/subscription		X									X				
Select * From MSFT_TransientRebootEvent																

Actually, Microsoft does not recommend any time and development investment in this feature due to the complexity of the current correlation implementation and some upcoming architectural changes regarding the future of the Windows management. Microsoft does not plan to enhance and support these WMI providers in the future and plans to provide a new correlation architecture, which will be .NET based with the next version of its operating system. You can check Chapter 5, section 5.8, “A Look into the Future of WMI Scripting,” for a view of the WMI Scripting future.

Therefore, we will not cover these providers and their related classes. However, since it is available in Windows XP, for greater details it is recommended that you refer to the platform SDK. For your information, Table 3.84 contains the list of correlation providers with their related classes as is right after installation.

3.10 Summary

In this chapter, we continued to use the same scripting techniques (enumeration, WQL queries, sink routines, etc.) as previously. However, at this stage, we clearly realize that the challenge is not limited to knowledge of the WMI Scripting API with the CIM repository classes. The challenge expands to the knowledge of the WMI provider capabilities with their classes, which determines the features available (i.e., instance provider versus event provider). In this case, the main concern is to understand the underlying technology the providers are managing. The WMI *SNMP* providers are a good example of providers that require a fair understanding of the SNMP MIBs before scripting on top of the WMI classes representing SNMP information. Moreover, if some private SNMP information must be added to the CIM repository, the knowledge of the managed device from an SNMP point of view is determinant.

Although the real-world managed entities vary throughout the chapter, the scripting technique remains a constant whatever the managed component is. This is exactly where we get the benefit of the abstraction made by WMI and the CIM repository. However, the management information nature, its interpretation, and its representation become the focus of the WMI development.

Now that we have a fair understanding of the core WMI providers under Windows Server 2003, there is still one aspect of WMI that we haven't yet examined: the WMI Security scripting. This is the purpose of the next chapter.

3.11 Useful Internet URLs

Windows Installer provider:

<http://www.microsoft.com/downloads/release.asp?releaseid=32832>

SNMP providers for Windows NT 4.0:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=f8130806-2589-46b3-b472-26b816776f31&DisplayLang=en>

RFC1213:

<http://www.faqs.org/rfcs/rfc1213.html>

RFC1215:

<http://www.faqs.org/rfcs/rfc1215.html>

Cisco Private MIB files:

<http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml>

Cisco MIB OID numbers translation:

<http://jaguar.ir.miami.edu/%7Emarcus/snmptrans.html>

Security Descriptor String Format

http://msdn.microsoft.com/library/en-us/security/Security/security_descriptor_string_format.asp