4

Scripting with WMI

4.1 Objective

In previous chapters, we examined the WMI architecture, WMI-related tools, and WQL. We now have the necessary knowledge to start the discussion of WMI scripting. Scripting on top of WMI implies the use of WMI COM objects. A script uses two types of objects: the objects obtained from the various class definitions located in the CIM repository (instances) and the objects exposed by the WMI scripting API (COM objects). As previously mentioned, in order to script on top of a COM abstraction layer, it is important to know how to navigate in the object model. To script on top of WMI, it is important to master both object models: the CIM repository object model and the WMI COM API object model. In Chapter 2, we focused on the tools that help understand and discover the CIM repository. In this chapter and the next, we focus on the WMI scripting API and show how this API can help to instantiate objects from the CIM repository and retrieve information about the CIM classes. We also find out how to work with real-world managed objects via this API by using CIM representations. The WMI scripting API is a collection of COM objects usable from most programming languages (such as Visual Basic or C++) and scripting languages (such as VBScript or JScript). In order to use the WMI COM objects they must first be instantiated. In addition, the CIM repository classes must also be instantiated as objects in order to reference the manageable entities of the real world. We are now in front of two instantiation types: the instantiation of the WMI scripting COM objects and the instantiation of the manageable entities. Both object types must be located in the system or in the CIM repository based on their nature (from the WMI scripting COM object collection or from the CIM repository definitions). In this chapter, we discover the techniques for using WMI scripting objects

with real-world manageable entities and examine the functionalities offered by these objects.

4.2 The WMI object path

To address manageable entities from the real world as defined in the CIM repository, it is necessary to locate their definitions (classes) in the CIM repository. The object path uniquely identifies the definitions made in the CIM repository. This path is used to locate three things:

1. A WMI namespace on a server
2. The classes defined in a particular namespace
3. The instances of a class in particular namespace.

The WMI object path can be compared with the concept of the universal resource locator (URL) used for the Internet. Both are hierarchical and use a number of elements in their structures, although the real representation does not look the same. The number of elements is determined by the nature of the object to locate. We can clearly distinguish three types of WMI object path:

1. **The namespace object path:** This path is used to locate a particular namespace in the CIM repository on a server. It uses two elements: the server name and the CIM repository namespace name. For example, the following string is a valid namespace object path:

   ```
   \\NET-DPEN6400A\Root\CIMv2
   ```

   where the server is NET-DPEN6400A and the namespace is **Root\CIMv2**. The generic representation of a namespace object path is

   ```
   \\Server\Namespace
   ```

   or

   ```
   //Server/Namespace
   ```

   or

   ```
   \\.\Namespace
   ```

   if the local server is referenced.

2. **The class object path:** This path is used to locate a particular class in a namespace on a server. It uses three elements: the server

name, the namespace, and the class name. For example, the following string is a valid class object path:

```
\\NET-DPEN6400A\Root\CIMv2\Win32_Service
```

where the server is NET-DPEN6400A, the namespace **Root\CIMv2**, and the class is *Win32_Service*. The generic representation of a class object path is:

```
\\Server\Namespace\ClassName
```

and follows the same variation as before.

3. **The instance object path:** This path is used to locate an instance of a particular class in a namespace on a server. It uses four elements: the server name, the namespace, and the instance name identified with the key property as defined by the class template. For example, the following string is a valid instance object path:

```
\\NET-DPEN6400A\Root\CIMv2\Win32_Service.Name="SNMP"
```

where the server is NET-DPEN6400A, the namespace **Root\CIMv2**, and the instance is the SNMP service of the *Win32_Service* class referenced by its key property, *name*. The generic representation of an instance object path is

```
\\Server\Namespace\ClassName.KeyName="KeyValue"
```

4.3 The WMI moniker

A moniker is a string (also called a display name) that provides a location and an identity to an object that must be instantiated in a script. This term comes from the COM object programming and represents another programmatic identifier (called ProgID) that maps to a class identifier (called CLSID). This CLSID points to a COM object implemented as a .dll containing the desired object. To define the location and the identity of an object, the moniker uses a syntax that is specific to any implementation. The WMI moniker is a string that combines some WMI-specific information with a WMI object path. That string is used to access information exposed by WMI. The specific information of a WMI moniker contains:

- A namespace (WinMgmts: for WMI)
- Some optional localization data
- Some optional security settings used for the object instantiation

The string "WinMgmts:" is not case sensitive and is dedicated to refer to WMI monikers. This will make the system point to the WBEMDISP.Dll located in %SystemRoot\System32\WBEM directory that is the .dll containing the WMI scripting COM API. Let's take a sample. In the previous paragraph, we saw an instance object path for the SNMP service. The moniker used to instantiate this *Win32_Service* instance in a script is

```
WinMgmts:\\NET-DPEN6400A\Root\CIMv2:Win32_Service.Name="SNMP"
```

To access this object instance in a script, the code can be written in VBScript as follows:

```
Set objWMIInstance = GetObject("WinMgmts:\\NET-DPEN6400A\Root\CIMv2:Win32_Service.name=""SNMP""")
```

You will notice the double quotes to respect the VBScript syntax. To ease reading this in VBScript, it can be changed to

```
Set objWMIInstance = GetObject("WinMgmts:\\NET-DPEN6400A\Root\CIMv2:Win32_Service.name='SNMP'")
```

In JScript, the backslashes and the quotes must be escaped (with a supplemental backslash [\]); therefore, the line will be

```
var objWMIInstance = GetObject("winmgmts:\\\\NET-DPEN6400A\\Root\\CIMv2:Win32_Service.name=\"SNMP\"");
```

Or, to suppress the escape sequence of the backslashes and the quotes to make the line more readable, it will be

```
var objWMIInstance = GetObject("winmgmts://NET-DPEN6400A/Root/CIMv2:Win32_Service.name='SNMP'");
```

or

```
var objWMIInstance = GetObject('winmgmts://NET-DPEN6400A/Root/CIMv2:Win32_Service.name="SNMP"');
```

4.3.1 Getting the SWbemObject with the moniker

Many things can be said about the previous moniker samples. The first thing to note is the nature of the object obtained and the fact that it is stored in the **objWMIInstance** variable. As this moniker points to an instance object path, it makes sense that we obtain an object instance. This object corresponds to the **SWbemObject** object as defined in the WMI scripting object model. We revisit this object later in Section 4.4.2. This object exposes properties and methods associated with the SNMP service provided by the *Win32_Service* class defined in the CIM repository. The list of properties and methods provided by the *Win32_Service* class is shown in Table 4.1.

Table 4.1 *The Win32_Service Properties and Methods*

Properties	AcceptPause	**Boolean** Read-only Indicates whether the service can be paused.
	AcceptStop	**Boolean** Read-only Indicates whether the service can be stopped.
	Caption	**string** Read-only Short description (one-line string) of the object.
	CheckPoint	**uint32** Read-only Value that the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation. For example, the service should increment this value as it completes each step of its initialization when it is starting up. The user interface program that invoked the operation on the service uses this value to track the progress of the service during a lengthy operation. This value is not valid and should be zero when the service does not have a start, stop, pause, or continue operation pending.
	CreationClassName	**string** Read-only Name of the first concrete class to appear in the inheritance chain used in the creation of an instance. When used with the other key properties of the class, the property allows all instances of this class and its subclasses to be uniquely identified.
	Description	**string** Read-only Description of the object.

Table 4.1 *The Win32_Service Properties and Methods (continued)*

Properties (cont'd.)	DesktopInteract	**Boolean** Read-only Indicates whether the service can create or communicate with windows on the desktop.
	DisplayName	**string** Read-only Display name of the service. This string has a maximum length of 256 characters. The name is case-preserved in the Service Control Manager; however, **DisplayName** comparisons are always case-insensitive. Constraints: Accepts the same value as the **Name** property.
	ErrorControl	**string** Read-only Severity of the error if this service fails to start during startup. The value indicates the action taken by the startup program if failure occurs. All errors are logged by the computer system.
	ExitCode	**uint32** Read-only Win32 error code defining any problems encountered in starting or stopping the service. This property is set to ERROR_SERVICE_SPECIFIC_ERROR (1066) when the error is unique to the service represented by this class, and information about the error is available in the **ServiceSpecificExitCode** property. The service sets this value to NO_ERROR when running, and again upon normal termination.
	InstallDate	**datetime** Read-only Indicates when the object was installed. A lack of a value does not indicate that the object is not installed.
	Name	**string** Read-only Uniquely identifies the service and provides an indication of the functionality that is managed. This functionality is described in more detail in the object's **Description** property.

Table 4.1 *The Win32_Service Properties and Methods (continued)*

Properties (cont'd.)	PathName	**string** Read-only Fully qualified path to the service binary file that implements the service.
	ProcessId	**uint32** Read-only Process identifier of the service.
	ServiceSpecificExitCode	**uint32** Read-only Service-specific error code for errors that occur while the service is either starting or stopping. The exit codes are defined by the service represented by this class. This value is only set when the ExitCode property value is ERROR_SERVICE_SPECIFIC_ERROR, 1066.
	ServiceType	**string** Read-only Type of service provided to calling processes.
	Started	**Boolean** Read-only Indicates whether the service has been started.
	StartMode	**string** Read-only Start mode of the Win32 base service.

Table 4.1 *The Win32_Service Properties and Methods (continued)*

Properties (cont'd.)	StartName	**string** Read-only Account name under which the service runs. Depending on the service type, the account name may be in the form of DomainName\Username. The service process will be logged using one of these two forms when it runs. If the account belongs to the built-in domain, .\Username can be specified. If NULL is specified, the service will be logged on as the LocalSystem account. For kernel or system level drivers, StartName contains the driver object name (that is, \FileSystem\Rdr or \Driver\Xns) which the input and output (I/O) system uses to load the device driver. Additionally, if NULL is specified, the driver runs with a default object name created by the I/O system based on the service name.
	State	**string** Read-only Current state of the base service.
	Status	**string** Read-only Current status of the object. Various operational and nonoperational statuses can be defined. Operational statuses include OK, Degraded, and Pred Fail (an element, such as a SMART-enabled hard drive, may be functioning properly but predicting a failure in the near future). Nonoperational statuses include Error,Starting, Stopping, and Service. The latter, Service, could apply during mirror-resilvering of a disk, reload of a user permissions list, or other administrative work. Not all such work is on-line, yet the managed element is neither OK nor in one of the other states.
	SystemCreationClassName	**string** Read-only Type name of the system that hosts this service.
	SystemName	**string** Read-only Name of the system that hosts this service.

Table 4.1 *The Win32_Service Properties and Methods (continued)*

Properties (cont'd.)	TagId	**uint32** Read-only Unique tag value for this service in the group. A value of 0 indicates that the service has not been assigned a tag. A tag can be used for ordering service startup within a load order group by specifying a tag order vector in the registry located at: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\GroupOrderList. Tags are only evaluated for kernel driver and file system driver start-type services that have boot or system start modes.
	WaitHint	**uint32** Read-only Estimated time required (in milliseconds) for a pending start, stop, pause, or continue operation. After the specified amount of time has elapsed, the service makes its next call to the **SetServiceStatus** method with either an incremented **CheckPoint** value or a change in **CurrentState**. If the amount of time specified by **WaitHint** passes, and **CheckPoint** has not been incremented, or **CurrentState** has not changed, the service control manager or service control program assumes that an error has occurred.
Methods	StartService	Class method that attempts to place the service into its startup state
	StopService	Class method that places the service in the stopped state
	PauseService	Class method that attempts to place the service in the paused state
	ResumeService	Class method that attempts to place the service in the resumed state
	InterrogateService	Class method that requests that the service update its state to the service manager
	UserControlService	Class method that attempts to send a user-defined control code to a service
	Create	Class method that creates a new service
	Change	Class method that modifies a service
	ChangeStartMode	Class method that modifies the start mode of a service
	Delete	Class method that deletes an existing service

If the script runs on the local server, the server name can be skipped in the moniker. In this case, the line is

```
Set objWMIInstance = GetObject("WinMgmts:\\.\Root\CIMv2:Win32_Service.name='SNMP'")
```

or

```
Set objWMIInstance = GetObject("WinMgmts:\Root\CIMv2:Win32_Service.name='SNMP'")
```

or

```
Set objWMIInstance = GetObject("WinMgmts:Root\CIMv2:Win32_Service.name='SNMP'")
```

Next, the scripting API uses the **Root\CIMv2** namespace by default. This setting is modifiable via the Computer Management MMC by editing the advanced properties of the WMI Control snap-in located in the Services and Applications.

As the **Root\CIMv2** namespace is the same as the default namespace, the line can be coded as follows:

```
Set objWMIInstance = GetObject("WinMgmts::Win32_Service.name='SNMP'")
```

or

```
Set objWMIInstance = GetObject("WinMgmts:Win32_Service.name='SNMP'")
```

Because the *name* property is defined as the only key in the *Win32_Service* class, the line can be coded as follows:

```
Set objWMIInstance = GetObject("WinMgmts:Win32_Service='SNMP'")
```

In all variations, the returned object is an instance of the SNMP service, and the COM nature of the **objWMIInstance** variable comes from the COM scriptable object called **SWbemObject**. Now, if the equality test of the line is removed as follows:

```
Set objWMIInstance = GetObject("WinMgmts:Win32_Service")
```

the returned object is not a real-world manageable object instance any more but an instance of the *Win32_Service* class. What is the difference? The difference is very small from a coding point of view, but very important! In the previous case, the instance returned pointed to the SNMP service itself, because the object path was a WMI instance object path. In the second case, the returned instance points to the class definition of the *Win32_Service* itself because the object path is a WMI class object path. Let's make a small test by using the following lines of code:

```
Set objWMIInstance = GetObject("WinMgmts:Win32_Service='SNMP'")
WScript.Echo objWMIInstance.Name
```

If this script runs, it echoes the name of the service: SNMP. Now, if the following piece of code is used instead:

```
Set objWMIInstance = GetObject("WinMgmts:Win32_Service")
WScript.Echo objWMIInstance.Name
```

it echoes a null. Why? Simply because the value associated with the name property for the *Win32_Service* class is empty. This can be easily checked using the **WMI CIM Studio** by examining the class definition of the *Win32_Service*. You will see that the value associated with the *name* key property is empty in the right pane containing the column names, column types, and column values. To complete the test, you can eventually set a value in this row and run the prior lines of code to see the change. In this case, you will see the value you typed in the class definition for the *name* key property (to clear the typed value, you must right-click on the value and select "set to <empty>"). This small test practically demonstrates the difference between a class instance and a real-world manageable object instance. In both cases, the script creates an instance but one represents the CIM class from the CIM repository and the other represents the real-world manageable entity represented by the CIM class and located with the object path contained in the moniker.

4.3.2 Getting the SWbemServices with the moniker

Using the same sample line of code:

```
Set objWMIInstance = GetObject("WinMgmts:Root\CIMv2:Win32_Service.name='SNMP'")
```

if we change the WMI object path to

```
Set objWMIServices = GetObject("WinMgmts:Root\CIMv2")
```

the code will open a namespace because the given path is a namespace object path. The obtained object is a COM scriptable object called **SWbemServices**. In this case the moniker is not changing from a real-world manageable object to an object representing a class from the CIM repository; it is changing the nature of the returned COM object. This implies that the object exposes other properties and methods. However, it is still possible, by using some of its methods, to retrieve a WMI instance representing a class from the CIM repository or a real-world manageable object. We revisit this object in Section 4.4.1. We also see that the WMI scripting API offers objects that can be used as an alternative to the moniker. But again, this will be covered later. Let's get back to the moniker. There are still a few things to say about this particular string.

As we can see, the way the moniker is coded influences the object type instantiated in the script. Besides the object instantiation at the script level, the composition of the moniker is directly related to the CIM repository. This is another reason why it is very important to understand and gather knowledge about the CIM repository object model.

4.3.3 The security settings of the moniker

Another feature of the moniker with the WMI object path is that it may contain security settings. These security settings correspond to the DCOM security settings visible with the **DCOMCNFG.exe** tool under Windows NT 4.0 and Windows 2000 or with the Component Services MMC snap-in under Windows XP and Windows.NET Server. There are three types of DCOM security settings:

1. **Authentication level:** As DCOM is used behind the scene, it is also possible to specify the DCOM authentication level to use. The authentication level can have seven different levels: *Default* (uses the default Windows authentication setting), *None*, *Connect*, *Call*, *Pkt*, *PktIntegrity*, and *PktPrivacy*. To specify the authentication level in the moniker, the moniker must be changed to

```
WinMgmts:{AuthenticationLevel=default}!Root\CIMv2:Win32_Service.name='SNMP'
```

   Refer to Table 4.2 for the list of authentication levels and their related meanings.

2. **Impersonation level:** As WMI uses DCOM, it is possible that WMI must authenticate on behalf of the user to some other COM objects located in local and remote systems. In this case, the WMI providers may refuse to perform certain operations, or they may return an incomplete set of information because the impersonation level is not sufficient. By default, WMI allows DCOM objects to use the credentials of the user. But some situations may require that DCOM objects use anonymous authentication or query the credentials of the user. The impersonation level determines how WMI must behave in regard to the user credentials. The impersonation level can have four different types: *Anonymous*, *Identify*, *Impersonate* (the default), and *Delegate*. To specify the impersonation level in the moniker, the previous string

```
WinMgmts:Root\CIMv2:Win32_Service.name='SNMP'
```

   must be changed to

```
WinMgmts:{impersonationLevel=impersonate}!Root\CIMv2:Win32_Service.name='SNMP'
```

Table 4.2 *The WMI authentication levels*

Moniker Name	Scripting API Contants	Value	Description
Default	wbemAuthenticationLevelDefault	0	WMI uses the default Windows Authentication setting
None	wbemAuthenticationLevelNone	1	Uses no authentication
Connect	WbemAuthenticationLevelConnect	2	Authenticates the credentials of the client only when the client establishes a relationship with the server
Call	WbemAuthenticationLevelCall	3	Authenticates only at the beginning of each call when the server receives the request
Pkt	WbemAuthenticationLevelPkt	4	Authenticates that all data received is from the expected client
PktIntegrity	WbemAuthenticationLevelPktIntegrity	5	Authenticates and verifies that none of the data transferred between client and server has been modified
PktPrivacy	WbemAuthenticationLevelPktPrivacy	6	Authenticates all previous impersonation levels and encrypts the argument value of each remote procedure call

By default, the impersonation level under Windows NT 4.0, Windows 2000, Windows XP, and Windows.NET Server is set in the registry to *Impersonate* (see Figure 4.1). Earlier versions of WMI for Windows NT 4.0 installed with WMI 1.1 used an

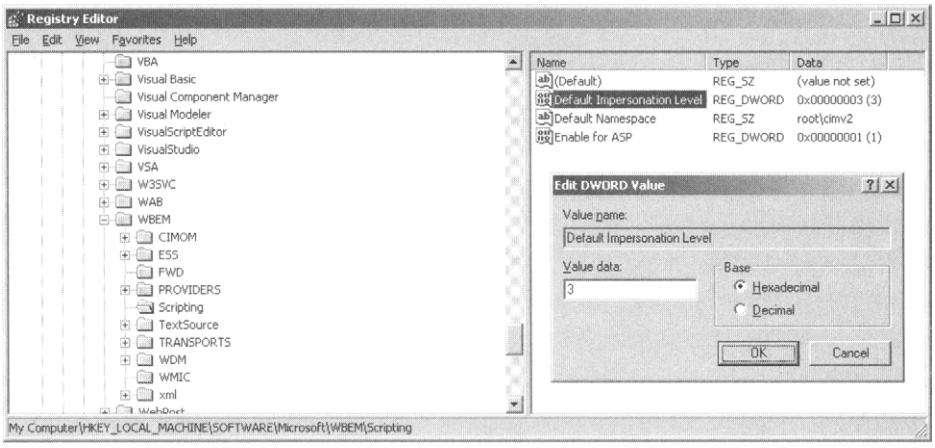


Figure 4.1 *The default impersonation level.*

impersonation level set to *Identify*. This implies that this setting is not required any more in the moniker. The default impersonation level is sufficient to perform most WMI operations.

However, it is a good practice to specify the impersonation level from the script to avoid any side effects due to the default value modification. Refer to Table 4.3 for the list of impersonation levels and their related meanings.

3. **Privileges:** To perform some WMI operations, it is sometimes required to provide some privileges. For instance, with WMI, it is possible to shut down a system or read the content of the NT Security Event Log. Without mentioning any privileges, these operations will fail. To specify privileges in the moniker, the moniker must be changed to

```
WinMgmts:{impersonationLevel=impersonate,(RemoteShutdown)}!Root\CIMv2:Win32_Service.name='SNMP'
```

Alternatively, it is possible to reset privileges. In this case, use the following moniker:

```
WinMgmts:{impersonationLevel=impersonate,(!RemoteShutdown)}!Root\CIMv2:Win32_Service.name='SNMP'
```

The "!" provides the negation of the privilege. Of course, the moniker can also include the privilege only:

```
WinMgmts:{(RemoteShutdown)}!Root\CIMv2:Win32_Service.name='SNMP'
```

Table 4.3 *The WMI Impersonation Levels*

Moniker Name	Scripting API Contants	Value	Description
Anonymous	wbemImpersonationLevelAnonymous	1	Hides the credentials of the caller. Calls to WMI may fail with this impersonation level.
Identify	wbemImpersonationLevelIdentify	2	Allows objects to query the credentials of the caller. Calls to WMI may fail with this impersonation level.
Impersonate	wbemImpersonationLevelImpersonate	3	Allows objects to use the credentials of the caller. This is the recommended impersonation level for WMI Scripting API calls.
Delegate	wbemImpersonationLevelDelegate	4	Allows objects to permit other objects to use the credentials of the caller. This impersonation, which will work with WMI Scripting API calls but may constitute an unnecessary security risk, is supported only under Windows_2000.

If several privileges must be set, the moniker will look as follows:

```
WinMgmts:{(RemoteShutdown, Security)}!Root\CIMv2:Win32_Service.name='SNMP'
```

Refer to Table 4.4 for the list of privilege levels and their related meanings.

When working with privileges, it is important to consider the Windows platform used. Under Windows 2000 and Windows.NET Server, privileges can be set or revoked at any time. Under Windows NT 4.0, the privileges must be set at connection time with the WMI system or when using the moniker string. Under Windows 95, Windows 98, and Windows Millennium, privileges can be set, but they do not have any effect.

The authentication type can also be specified with the authority parameter. In this case, the moniker not only specifies the authentication protocol to use (NTLM or Kerberos), it also mentions the authority. For a Kerberos authentication, the moniker is

```
WinMgmts:{authority=kerberos:MyDomain\RemoteServer}!\\RemoteServer\Root\CIMv2:Win32_Service.name='SNMP'
```

For NTLM, the moniker is

```
WinMgmts:{authority=ntlmdomain:MyDomain}!\\RemoteServer\Root\CIMv2:Win32_Service.name='SNMP'
```

The authority parameter can be used only with remote WMI connections. If it is used to connect locally, the operation will fail. This is the reason for the *RemoteServer* name addition in the WMI object path.

Even if it is possible to simplify the moniker string by assuming some system defaults, it is recommended to specify the defaults to avoid any surprises when running scripts with other systems. For instance, the minimal moniker string with the namespace, the impersonation level, and the authentication level to obtain a class instance is

```
WinMgmts:{impersonationLevel=impersonate,AuthenticationLevel=default}!Root\CIMv2:Win32_Service
```

4.3.4 The localization string of the moniker

The last element of information included in a moniker is the localization string. In Chapter 2, we saw that some namespaces have a child namespace containing a language-specific version of the classes. When opening a language-specific class, WMI combines the language-neutral definition with the language-specific definition to provide the localized version of the class. To obtain the localized version of an object, information must be added to

Table 4.4 *The WMI Privileges*

Moniker Name	Privilege Sstring	Scripting API Contants	Value	Description
CreateToken	SeCreateTokenPrivilege	wbemPrivilegeCreateToken	1	Required to create a primary token.
PrimaryToken	SeAssignPrimaryTokenPrivilege	wbemPrivilegePrimaryToken	2	Required to assign the primary token of a process.
LockMemory	SeLockMemoryPrivilege	wbemPrivilegeLockMemory	3	Required to lock physical pages in memory.
IncreaseQuota	SeIncreaseQuotaPrivilege	wbemPrivilegeIncreaseQuota	4	Required to increase the quota assigned to a process.
MachineAccount	SeMachineAccountPrivilege	wbemPrivilegeMachineAccount	5	Required to create a machine account.
Tcb	SeTcbPrivilege	wbemPrivilegeTcb	6	Identifies its holder as part of the trusted computer base. Some trusted, protected subsystems are granted this privilege.
Security	SeSecurityPrivilege	wbemPrivilegeSecurity	7	Required to perform a number of security-related functions, such as controlling and viewing audit messages. This privilege identifies its holder as a security operator.
TakeOwnership	SeTakeOwnershipPrivilege	wbemPrivilegeTakeOwnership	8	Required to take ownership of an object without being granted discretionary access. This privilege allows the owner value to be set only to those values that the holder may legitimately assign as the owner of an object.
LoadDriver	SeLoadDriverPrivilege	wbemPrivilegeLoadDriver	9	Required to load or unload a device driver.
SystemProfile	SeSystemProfilePrivilege	wbemPrivilegeSystemProfile	10	Required to gather profiling information for the entire system.
Systemtime	SeSystemtimePrivilege	wbemPrivilegeSystemtime	11	Required to modify the system time.
ProfileSingleProcess	SeProfileSingleProcessPrivilege	wbemPrivilegeProfileSingleProcess	12	Required to gather profiling information for a single process.

Table 4.4 *The WMI Privileges (continued)*

Moniker Name	Privilege Sstring	Scripting API Contants	Value	Description
IncreaseBasePriority	SeIncreaseBasePriorityPrivilege	wbemPrivilegeIncreaseBasePriority	13	Required to increase the base priority of a process.
CreatePagefile	SeCreatePagefilePrivilege	wbemPrivilegeCreatePagefile	14	Required to create a paging file.
CreatePermanent	SeCreatePermanentPrivilege	wbemPrivilegeCreatePermanent	15	Required to create a permanent object.
Backup	SeBackupPrivilege	wbemPrivilegeBackup	16	Required to perform backup operations.
Restore	SeRestorePrivilege	wbemPrivilegeRestore	17	Required to perform restore operations. This privilege enables you to set any valid user or group SID as the owner of an object.
Shutdown	SeShutdownPrivilege	wbemPrivilegeShutdown	18	Required to shut down a local system.
Debug	SeDebugPrivilege	wbemPrivilegeDebug	19	Required to debug a process.
Audit	SeAuditPrivilege	wbemPrivilegeAudit	20	Required to generate audit-log entries.
SystemEnvironment	SeSystemEnvironmentPrivilege	wbemPrivilegeSystemEnvironment	21	Required to modify the nonvolatile RAM of systems that use this type of memory to store configuration information.
ChangeNotify	SeChangeNotifyPrivilege	wbemPrivilegeChangeNotify	22	causes the system to skip all traversal access checks. It is enabled by default for all users.
RemoteShutdown	SeRemoteShutdownPrivilege	wbemPrivilegeRemoteShutdown	23	Required to shut down a system using a network request.
Undock	SeUndockPrivilege	wbemPrivilegeUndock	24	Required to remove computer from docking station.
SyncAgent	SeSyncAgentPrivilege	wbemPrivilegeSyncAgent	25	Required to synchronize directory service data.
EnableDelegation	SeEnableDelegationPrivilege	wbemPrivilegeEnableDelegation	26	Required to enable computer and user accounts to be trusted for delegation.

the moniker to specify that the localized version of the class must be used. In this case, the moniker becomes

```
WinMgmts:[locale=ms_409]!Root\CIMv2:Win32_Service='SNMP'
```

where the **ms_409** specifies the U.S. language localized version.

4.3.5 A first script sample using the WMI moniker

Sample 4.1 shows a VBScript sample listing the properties of a *Win32_Service* instance called "SNMP." The moniker has been adapted to put the miscellaneous parameters in VBScript constants (lines 18 to 28). From lines 30 to 55, the script displays the properties of the WMI object instance set at line 25 with the moniker.

Sample 4.1 *A VBScript listing the Win32_Service instance properties*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <script language="VBscript">
14:    <![CDATA[
..:
18:    Const cComputerName = "LocalHost"
19:    Const cWMINameSpace = "root/cimv2"
20:    Const cWMIClass     = "Win32_Service"
21:    Const cWMIInstance  = "SNMP"
..:
25:    Set objWMIInstance = GetObject("winmgmts:{impersonationLevel=impersonate}!//" & _
26:                                   cComputerName & "/" & _
27:                                   cWMINameSpace & ":" & _
28:                                   cWMIClass & ".Name='" & cWMIInstance & "'")
29:
30:    WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
31:    WScript.Echo "   AcceptPause=" & objWMIInstance.AcceptPause
32:    WScript.Echo "   AcceptStop=" & objWMIInstance.AcceptStop
33:    WScript.Echo "   Caption=" & objWMIInstance.Caption
34:    WScript.Echo "   CheckPoint=" & objWMIInstance.CheckPoint
35:    WScript.Echo "   CreationClassName=" & objWMIInstance.CreationClassName
36:    WScript.Echo "   Description=" & objWMIInstance.Description
37:    WScript.Echo "   DesktopInteract=" & objWMIInstance.DesktopInteract
38:    WScript.Echo "   DisplayName=" & objWMIInstance.DisplayName
39:    WScript.Echo "   ErrorControl=" & objWMIInstance.ErrorControl
40:    WScript.Echo "   ExitCode=" & objWMIInstance.ExitCode
41:    WScript.Echo "   InstallDate=" & objWMIInstance.InstallDate
42:    WScript.Echo "   Name=" & objWMIInstance.Name
43:    WScript.Echo "   PathName=" & objWMIInstance.PathName
44:    WScript.Echo "   ProcessId=" & objWMIInstance.ProcessId
45:    WScript.Echo "   ServiceSpecificExitCode=" & objWMIInstance.ServiceSpecificExitCode
46:    WScript.Echo "   ServiceType=" & objWMIInstance.ServiceType
47:    WScript.Echo "   Started=" & objWMIInstance.Started
48:    WScript.Echo "   StartMode=" & objWMIInstance.StartMode
```

```
49:    WScript.Echo "   StartName=" & objWMIInstance.StartName
50:    WScript.Echo "   State=" & objWMIInstance.State
51:    WScript.Echo "   Status=" & objWMIInstance.Status
52:    WScript.Echo "   SystemCreationClassName=" & objWMIInstance.SystemCreationClassName
53:    WScript.Echo "   SystemName=" & objWMIInstance.SystemName
54:    WScript.Echo "   TagId=" & objWMIInstance.TagId
55:    WScript.Echo "   WaitHint=" & objWMIInstance.WaitHint
..:
59:    ]]>
60:    </script>
61:  </job>
62:</packages
```

As we have seen in this section, the moniker offers many combinations. These combinations are summarized in generic notation as follows:

```
"WinMgmts:" securitySetting ["[" localeSetting "]"] ["!" objectPath]
Or,
"WinMgmts:" "[" localeSetting "]" ["!" objectPath]
Or,
"WinMgmts:" [objectPath]

Where: localeSetting : "locale" "=" localeID
|
Where: localeID      : a value of the form "ms_xxxx" where xxxx is a hex LCID value (e.g. "ms_409")
|
Where: objectPath    : a valid WMI Object Path
|
Where: securitySetting is equal to:
    |
    |  "{"  authAndImpersonSettings [ ","  privilegeOverrides] "}"
    |  Or,
    |  "{"  privilegeOverrides  "}"
    |
    Where: authAndImpersonSettings is equal to:
    |   |
    |   |  authenticationLevel
    |   |  Or,
    |   |  impersonationLevel
    |   |  Or,
    |   |  authority
    |   |  Or,
    |   |  authenticationLevel  ","  impersonationLevel [ ","  authority]
    |   |  Or,
    |   |  authenticationLevel  ","  authority [ ","  impersonationLevel]
    |   |  Or,
    |   |  impersonationLevel  ","  authenticationLevel [ ","  authority]
    |   |  Or,
    |   |  impersonationLevel  ","  authority [ ","  authenticationLevel]
    |   |  Or,
    |   |  authority  ","  impersonationLevel [ ","  authenticationLevel]
    |   |  Or,
    |   |  authority  ","  authenticationLevel [ ","  impersonationLevel]
    |   |
    |   Where: authority : "authority" "=" authorityValue
    |   |   |
    |   |   +-- Where: authorityValue : "kerberos:mydomain\RemoteServer" | "ntlmdomain:mydomain".
    |   |
    |   Where: authenticationLevel : "authenticationLevel" "=" authenticationValue
```

```
|   |   |
|   |   +-- Where: authenticationValue : "default" |
|   |                                    "none" |
|   |                                    "connect" |
|   |                                    "call" |
|   |                                    "pkt" |
|   |                                    "pktIntegrity" |
|   |                                    "pktPrivacy"
|   |
|   Where: impersonationLevel : "impersonationLevel" "=" impersonationValue
|       |
|       +-- Where: impersonationValue : "anonymous" |
|                                       "identify" |
|                                       "impersonate" |
|                                       "delegate"
|
Where: privilegeOverrides : "(" privileges ")"
    |
    +-- Where: privileges : privilege [ "," privilege [ "," privilege ]]*
            |
            +-- Where: privilege : ["!"] privilegeName
                |
                +-- Where: privilegeName is equal to:

                        "CreateToken"          | "PrimaryToken"     | "LockMemory" |
                        "IncreaseQuota"        | "MachineAccount"   | "Tcb" |
                        "Security"             | "TakeOwnership"    | "LoadDriver" |
                        "SystemProfile"        | "SystemTime"       | "ProfileSingleProcess" |
                        "IncreaseBasePriority" | "CreatePagefile"   | "CreatePermanent" |
                        "Backup"               | "Restore"          | "Shutdown" |
                        "Debug"                | "Audit"            | "SystemEnvironment" |
                        "ChangeNotify"         | "RemoteShutdown"   | "Udock" |
                        "SyncAgent"            | "EnableDelegation"
```

4.4 Exploring the WMI scripting API

In the previous paragraph, based on the moniker type provided, we saw that it is possible to obtain two different WMI objects:

1. **SWbemObject** with the moniker

   ```
   winmgmts://NET-DPEN6400A/Root/CIMv2:Win32_Service.name="SNMP"'
   ```

   or

   ```
   winmgmts://NET-DPEN6400A/Root/CIMv2:Win32_Service
   ```

2. **SWbemServices** with the moniker

   ```
   winmgmts://NET-DPEN6400A/Root/CIMv2
   ```

These two objects are part of the WMI scripting API. The WMI scripting API is a collection of COM objects enabling a script writer to read or write information and perform various operations with WMI and the real-

world manageable entities that the CIM repository represents. The WMI scripting object model exposes two different natures of objects that can be categorized as follows:

1. Single objects
2. Collections of objects

The nature of the object desired determines the way the object is instantiated. There are two different ways to instantiate objects with WMI:

1. Using a language statement to create an object with the COM ProgID. Based on the script language used, the script code will use a **CreateObject** for VBScript, a **new ActiveXObject** for JScript, or an XML <Object> tag in an ASP page or a Windows Script File.
2. Using the WMI moniker with the **GetObject** statement for VBScript or JScript.

The different object natures with their associated instantiation methods are summarized in Figure 4.2 as represented by the Microsoft SDK.

Symbols are used to differentiate the objects. In the following sections, we examine the WMI scripting API object model tree and explain the relationship between the various objects. The symbols will help to locate objects and determine how they can be instantiated. By using different small pieces of script, we will navigate and explore the WMI object model step by step.

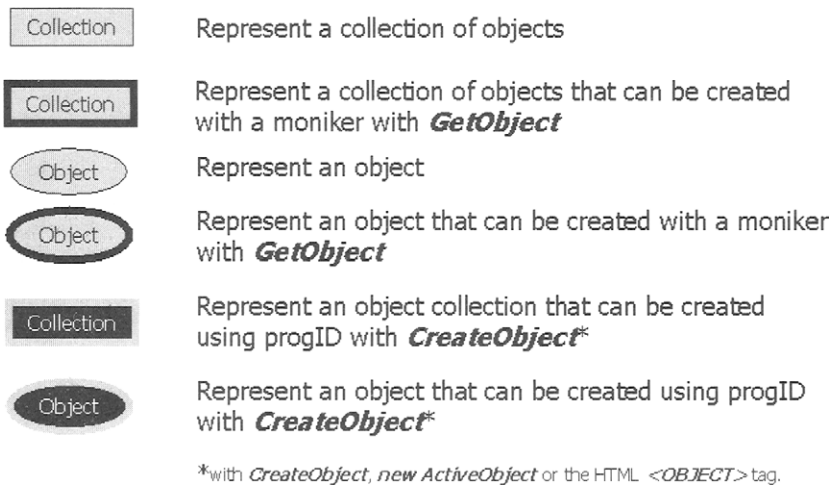


Figure 4.2 *Symbols used to represent the WMI object model.*

4.4.1 Establishing the WMI connection

To be connected to a real-world manageable entity, we must connect to the object representing that entity in the WMI object model. This represents the first step of the WMI object model discovery. The first part of the object model is represented in Figure 4.3. A circled number labels each object on the figure. This will ease the reference to and the positioning of the object in the model in the text that follows.

Any object class, object instance, or setting that can be instantiated with the moniker can also be instantiated with the scripting API. Every element that constitutes a moniker can also be represented with a WMI COM object part of the WMI scripting API. Let's start from the beginning by using a sample. When using the moniker:

```
winmgmts://NET-DPEN6400A/Root/CIMv2
```

we already have two things: a namespace and the server to connect to. By using the WMI scripting API and decomposing the moniker elements, it is

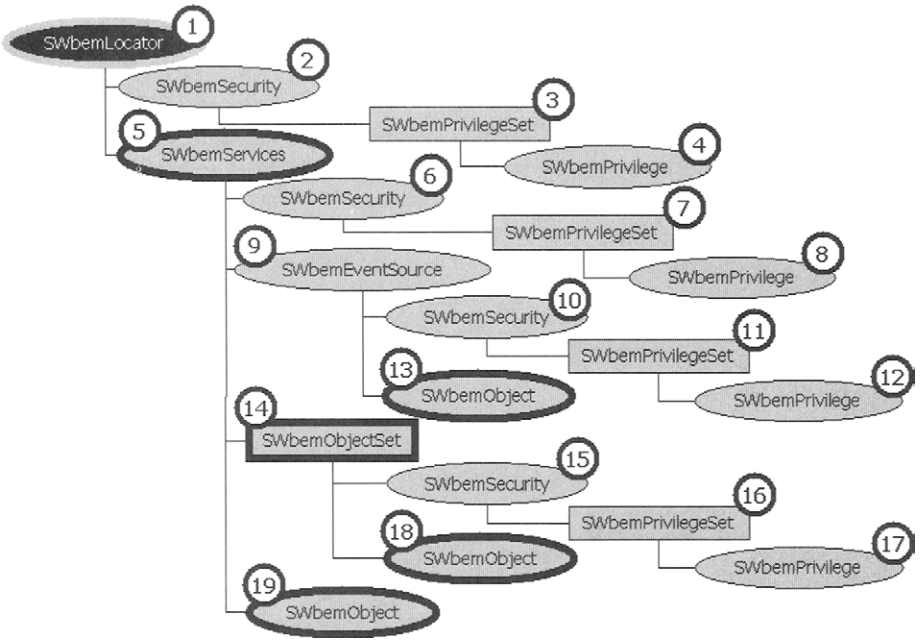


Figure 4.3 *The WMI object model (1/3).*

possible to perform the same operation. In other words, a script using this moniker

```
Set objWMIServices = GetObject ("winmgmts://NET-DPEN6400A/Root/CIMv2")
```

is equal to the following script:

```
 .:
 7:Set objWMILocator = CreateObject("WbemScripting.SWbemLocator")
..:
14:Set objWMIServices = objWMILocator.ConnectServer("NET-DPEN6400A", "Root\CIMv2", "", "")
```

The main difference is that there is no longer a moniker string, but the **objWMIServices** is obtained with the WMI scripting API by creating an **SWbemLocator** object (line 7) and using its *ConnectServer* method (line 14). The result of the method gives an **SWbemServices** object (line 14). The **SWbemLocator** is the first object to be created and is represented by the number 1 in Figure 4.3.

One interesting aspect of the **SWbemLocator** object is the possibility to specify other credentials and a localization string (i.e., ms_409) in the parameters of its *ConnectServer* methods. If your script must specify credentials to establish the connection, there is no alternative to using the **SWbemLocator** object because this is not supported by the WMI moniker.

The **SWbemLocator** object properties and methods are given in Table 4.5. In the table we see that a security property is available. This property can be used to set the same security settings as those set when using the moniker. For instance, in the moniker samples we saw in Section 4.3, we set

Table 4.5 *The SWbemLocator Object*

SWbemLocator			
Properties	Security_		Used to read or change the security settings.
Methods	ConnectServer	[strServer = ""], [strNameSpace = ""], [strUser = ""], [strPassword = ""], [strLocale = ""], [strAuthority = ""], [intSecurityFlags = 0], [objwbemNamedValueSet = null]	Connects to WMI on the specified computer.

three different things: the impersonation level, the authentication method, and some privileges. In this case the moniker is as follows:

```
WinMgmts:{impersonationLevel=impersonate, AuthenticationLevel=default,
                                        (RemoteShutdown, Security)}!\\NET-DPEN6400A\Root\CIMv2
```

How do we set the security settings used in the moniker via the scripting API? By referencing the *security_* property of the **SWbemLocator** object, it is possible to retrieve the **SWbemSecurity** object. In this case, the script creates the object represented by the number 2 in Figure 4.3. The code is as follows:

```
..:
 7:Set objWMILocator = CreateObject("WbemScripting.SWbemLocator")
 8:Set objWMISecurity = objWMILocator.Security_
..:
14:Set objWMIServices = objWMILocator.ConnectServer("NET-DPEN6400A", "Root\CIMv2", "", "")
```

The **SWbemSecurity** object (line 8) exposes three properties (see Table 4.6) that correspond to the three security settings available from DCOM: authentication level, impersonation level, and the privileges.

Each of these settings can receive an assignment with a WMI constant from Table 4.2 for the authentication setting, Table 4.3 for the impersonation setting, and Table 4.4 for the privilege settings, respectively. The code is as follows:

```
 1:Const wbemImpersonationLevelImpersonate = 3
 2:Const wbemAuthenticationLevelDefault = 0
 3:
 4:Const wbemPrivilegeSecurity = 7
 5:Const wbemPrivilegeRemoteShutdown = 23
 6:
 7:Set objWMILocator = CreateObject("WbemScripting.SWbemLocator")
 8:Set objWMISecurity = objWMILocator.Security_
 9:objWMISecurity.AuthenticationLevel = wbemAuthenticationLevelDefault
10:objWMISecurity.ImpersonationLevel = wbemImpersonationLevelImpersonate
11:Set objWMIPrivilegeSet = objWMISecurity.Privileges
..:
14:Set objWMIServices = objWMILocator.ConnectServer("NET-DPEN6400A", "Root\CIMv2", "", "")
```

Table 4.6 *The SWbemSecurity Object*

SWbemSecurity		
Properties	AuthenticationLevel	Numeric value that defines the DCOM Authentication level assigned to this object. This setting determines how you protect information sent from WMI.
	ImpersonationLevel	Numeric value that defines the DCOM Impersonation level assigned to this object. This setting determines if processes owned by WMI can detect or use your security credentials when making calls to other processes.
	Privileges	An SWbemPrivilegeSet object that defines the Windows_NT/ Windows_2000 Privileges for this object.

Table 4.7 *The SWbemPrivilegeSet Object*

SWbemPrivilegeSet		
Properties	Count	The number of items in the collection.
Methods	Add intPrivilege, [boolIsEnabled = True]	Adds an SWbemPrivilege object to the SWbemPrivilegeSet collection using a WbemPrivilegeEnum constant.
	AddAsString strPrivilege, [boolIsEnabled = True]	Adds an SWbemPrivilege object to the SWbemPrivilegeSet collection using a Windows_NT/ Windows_2000 privilege string.
	Item intPrivilege	Retrieves an SWbemPrivilege object from the collection. This is the default method of this object.
	DeleteAll ()	Deletes all the privileges from the collection.
	Remove intPrivilege	Removes an SWbemPrivilege object from the collection.

As we can see, the privileges property from the **SWbemSecurity** object does not take a constant as parameter, but retrieves another object (line 11). The obtained object is called **SWbemPrivilegeSet** and exposes one property and several methods to set the privileges (see Table 4.7). In the next sample, lines 12 and 13 add two privileges to the **SWbemPrivilegeSet** collection. The **SWbemPrivilegeSet** object in the WMI object model is represented by the number 3 in Figure 4.3. So, the script becomes

```
 1:Const wbemImpersonationLevelImpersonate = 3
 2:Const wbemAuthenticationLevelDefault = 0
 3:
 4:Const wbemPrivilegeSecurity = 7
 5:Const wbemPrivilegeRemoteShutdown = 23
 6:
 7:Set objWMILocator = CreateObject("WbemScripting.SWbemLocator")
 8:Set objWMISecurity = objWMILocator.Security_
 9:objWMISecurity.AuthenticationLevel = wbemAuthenticationLevelDefault
10:objWMISecurity.ImpersonationLevel = wbemImpersonationLevelImpersonate
11:Set objWMIPrivilegeSet = objWMISecurity.Privileges
12:objWMIPrivilegeSet.Add wbemPrivilegeRemoteShutdown, True
13:objWMIPrivilegeSet.Add wbemPrivilegeSecurity, True
14:Set objWMIServices = objWMILocator.ConnectServer("NET-DPEN6400A", "Root\CIMv2", "", "")
```

Each privilege addition represents an **SWbemPrivilege** object in the collection contained in **SWbemPrivilegeSet** object. The **SWbemPrivilege** object is represented by the number 4 in Figure 4.3. We can see that **SWbemPrivilege** is a child object of **SWbemPrivilegeSet**, which is in turn a child object of the **SWbemSecurity** object. The final script, which uses the features of WSH to access the WMI Type Library and instantiate objects, is shown in Sample 4.2.

Sample 4.2 *Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege constants set at the SWbemLocator object level*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
..:
27:    Set objWMISecurity = objWMILocator.Security_
28:    objWMISecurity.AuthenticationLevel = wbemAuthenticationLevelDefault
29:    objWMISecurity.ImpersonationLevel = wbemImpersonationLevelImpersonate
30:    Set objWMIPrivilegeSet = objWMISecurity.Privileges
31:    objWMIPrivilegeSet.Add wbemPrivilegeRemoteShutdown, True
32:    objWMIPrivilegeSet.Add wbemPrivilegeSecurity, True
33:    Set objWMIServices = objWMILocator.ConnectServer("NET-DPEN6400A", "Root\CIMv2", "", "")
..:
39:    ]]>
40:    </script>
41:  </job>
42:</package>
```

In Sample 4.3, as an alternative to using constants for privileges (lines 31 and 32), the **SWbemSecurity** object also accepts privileges as an explicit string (see Table 4.4) by using the *AddAsString* method of the **SwbemPrivilegeSet** object.

Sample 4.3 *Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege strings set at the SWbemLocator object level*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
..:
27:    Set objWMISecurity = objWMILocator.Security_
28:    objWMISecurity.AuthenticationLevel = wbemAuthenticationLevelDefault
29:    objWMISecurity.ImpersonationLevel = wbemImpersonationLevelImpersonate
30:    Set objWMIPrivilegeSet = objWMISecurity.Privileges
31:    objWMIPrivilegeSet.AddAsString "SeRemoteShutdownPrivilege", True
```

```
32:    objWMIPrivilegeSet.AddAsString "SeSecurityPrivilege", True
33:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
..:
39:    ]]>
40:    </script>
41:  </job>
42:</package>
```

The last two samples use the WMI scripting API and are equivalent to the following moniker sample shown in Sample 4.4.

Sample 4.4 *Obtaining an SWbemServices object with the WMI moniker*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <script language="VBscript">
14:    <![CDATA[
..:
18:    Const cComputerName = "LocalHost"
19:    Const cWMINameSpace = "root/cimv2"
..:
23:    Set objWMIServices = GetObject("WinMgmts:{impersonationLevel=impersonate, " & _
24:                                   "AuthenticationLevel=default, " & _
25:                                   "(RemoteShutdown, Security)}!\\" & _
26:                                   cComputerName & "\" & cWMINameSpace)
27:    Set objWMIServices = Nothing
28:    ]]>
29:    </script>
30:  </job>
31:</package>
```

As we can see the moniker version is much more compact, which is the biggest advantage of the moniker. One single line equals almost seven lines using the WMI scripting API. When using a moniker, the script starts directly at the level of the **SWbemServices** object (number 5 in Figure 4.3); when using the WMI scripting API, it requires the script to create the **SWBemLocator** object (number 1 in Figure 4.3) and **SWbemServices** (number 5 in Figure 4.3) with the associated security objects (numbers 2, 3, and 4 in Figure 4.3). Of course, if the script requires credentials to run in a different security context than the current user security context, the script must use the **SWbemLocator** object and the subsequent logic as shown in Sample 4.2 ("Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege constants set at the SWbemLocator object level").

Although both samples produce the same **SWbemServices** object, there is an important difference between Sample 4.3 ("Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege strings set

at the SWbemLocator object level") and Sample 4.4 ("Obtaining an SWbemServices object with the WMI moniker"). The difference resides in where the security is set. Sample 4.3 sets the security at the level of the **SWbemLocator** object (lines 27 to 32) and Sample 4.4 sets the security at the level of the **SWbemServices** objects. With the WMI scripting API, the security settings can be set at different object levels. The impersonation level, the authentication level, and the privileges can be set on the **SWbemServices**, **SWbemObject**, **SWbemObjectSet**, **SWbemObjectPath**, and **SWbemLocator** objects by setting their properties to the desired values.

A script can take advantage of this possibility, but there are some restrictions when running under Windows NT regarding privileges. Under Windows NT, it is mandatory to set the privileges at connection time because privilege changes can be made only on the process security token. As the process security token is created at connection time, all the security settings are set in Sample 4.3 at the level of the **SWbemLocator** object and in Sample 4.4 at the level of the moniker creating the **SWbemServices** object. Because the connection is created with the security settings when these two objects are created, the two samples are compatible with Windows NT. Under Windows 2000, Windows XP, and Windows.NET Server, privileges can be set or removed at any time. For example, Sample 4.3 can be changed to set the privileges after establishing the connection (see Sample 4.5).

Sample 4.5 *Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege strings set at the SWbemServices object level*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
..:
25:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
26:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
27:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
28:    objWMIServices.Security_.Privileges.AddAsString "SeRemoteShutdownPrivilege", True
29:    objWMIServices.Security_.Privileges.AddAsString "SeSecurityPrivilege", True
..:
33:    ]]>
34:    </script>
35:  </job>
36:</package>
```

In this sample, the privileges are set at the level of the **SWbemServices** object (lines 28 and 29) once the connection has been created (line 27). Valid under Windows 2000, Windows XP, or Windows.NET Server, this will not set the expected privileges under Windows NT because the privileges are set after establishing the connection (line 27).

The ability to set the security settings at the level of the **SWbemServices** object is represented by the numbers 6, 7, and 8 in the object model shown in Figure 4.3. You will also notice that it is possible to combine the object assignments in one line instead of creating an intermediate security object to assign authentication level (line 25), impersonation level (line 26), or privileges (lines 28 and 29). Once completed, the **SWbemServices** object obtained is the same as in previous samples (if running under Windows 2000, Windows XP, or Windows.NET Server).

4.4.2 Retrieving WMI objects

At this stage of the script, the WMI namespace is opened (i.e., **Root\CIMv2**). The **SWbemServices** object provides access to the namespace regardless of how the object was retrieved (moniker or scripting API). From this point, it is possible to instantiate a WMI class representing a real-world manageable object. An **SWbemObject** object in the script can represent this real-world manageable object. In the WMI object model (Figure 4.3) numbers 13, 18, and 19 locate the **SWbemObject** object. As we can see, different WMI scripting objects can produce the **SWbemObject** object based on the initial WMI object (the parent object), methods, and parameters used. The WMI scripting objects that can be used to create an **SWbemObject** are the **SWbemServices** object (number 5), the **SWbemObjectSet** object (number 14), or the **SWbemEventSource** object (number 9). Whatever the technique used to retrieve the **SWbemObject** object, at this stage of the discovery the retrieved **SWbemObject** object is always the same. If we look at the last script samples (Samples 4.2 to 4.5), we are at the level of the **SWbemServices** object and use its exposed methods to create an **SWbemObject** object.

As a first step, we examine how an **SWbemObject** object can be retrieved synchronously. "Synchronously" means that the code retrieving the **SWbemObject** waits for the retrieval operation to complete. In the next chapter we see that it is possible to retrieve an **SWbemObject** asynchronously, which means that the retrieval process is executed in parallel and the script does not wait for the operation to complete. Synchronously or asynchronously, the object method retrieval produces an **SWbemObject** object or an **SWbemObjectSet** object from **SWbemServices** object (number 5).

The **SWbemEventSource** object also produces an **SWbemObject** object, but only in the context of a WMI event. The WMI events scripting technique is examined in Chapter 6 of this book.

Now, what type of WMI objects can we obtain? The nature of the retrieved object can be a single class (in which case it is a single **SWbemObject** object as represented by number 19 in Figure 4.3), a class collection (in which case it is a **SWbemObjectSet** collection object as represented by number 14 in Figure 4.3), an instance (again an **SWbemObject** object), or an instance collection (again an **SWbemObjectSet** object). In addition to retrieving classes or instances from a WMI class (i.e., *Win32_Service*), it is also possible to retrieve a collection of classes or instances that are associated, referenced, or inherited from the specified class name or instance name. To retrieve a WMI object, an **SWbemService** object method can be used. It is also possible to use a WQL query entered in a specific **SWbemService** object method.

The properties and methods of the **SWbemServices** object are shown in Table 4.8. The content of this table will be examined in the rest of this section.

To retrieve an **SWbemObject** object, many methods exposed by **SWbemServices** can be used, but the easiest one to start with is the *Get* method. The next line of code is:

```
Set objWMIInstance = objWMIServices.Get ("Win32_Service")
```

This line will retrieve the class instance of *Win32_Service* from the CIM repository. If the line is changed as follows:

```
Set objWMIInstance = objWMIServices.Get ("Win32_Service='SNMP'")
```

the script line retrieves an instance of the SNMP service. Note that you can combine the moniker creating the **SWbemService** with its associated method. If Sample 4.4 ("Obtaining an SWbemServices object with the WMI moniker") is modified, the line creating the moniker can be changed to

```
Set objWMIInstance = GetObject("WinMgmts:{impersonationLevel=impersonate, " & _
                               "AuthenticationLevel=default, " & _
                               "(RemoteShutdown, Security)}!\\" & _
                               cComputerName & "\" & cWMINameSpace).Get _
                               (cWMIClass & "='" & cWMIInstance & "'")
```

In such a case, the object returned by the combined use of the moniker and the **SWbemServices** *Get* method is also an instance of the SNMP service. Both ways are valid and return exactly the same object. The moniker creates an **SWbemServices** object, and, on the same line, the *Get* method is invoked. So, before diving deeper into the exploration of the **SWbemServices**

Table 4.8 *The SWbemServices Object*

SWbemServices (with SWbemServicesEx)		
Properties	Security_	Used to read or change the security settings.
Methods	AssociatorsOf strObjectPath, [strAssocClass = ""], [strResultClass = ""], [strResultRole = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredAssocQualifier = ""], [strRequiredQualifier = ""], [intFlags = wbemFlagReturnImmediately], [objwbemNamedValueSet = null]	Returns a collection of objects (classes or instances) that are associated with a specified object. This method performs the same function that the ASSOCIATORS OF WQL query performs.
	AssociatorsOfAsync objWbemSink, strObjectPath, [strAssocClass = ""], [strResultClass = ""], [strResultRole = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredAssocQualifier = ""], [strRequiredQualifier = ""], [intFlags = wbemFlagDontSendStatus], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously returns a collection of objects (classes or instances) that are associated with a specified object. This method performs the same function that the ASSOCIATORS OF WQL query performs.
	Delete strObjectPath, [intFlags = 0], [objWbemNamedValueSet = null]	Deletes an instance or class.

Table 4.8 *The SWbemServices Object (continued)*

SWbemServices (with SWbemServicesEx)		
Methods (cont'd.)	DeleteAsync [objWbemSink = null], strObjectPath, [intFlags = 0], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously deletes an instance or class.
	ExecMethod strObjectPath, strMethodName, [objWbemInParams = null], [intFlags = 0], [objWbemNamedValueSet = null]	Executes an object method.
	ExecMethodAsync objWbemSink, strObjectPath, strMethodName, [objWbemInParams = null], [intFlags = 0], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously executes an object method.
	ExecNotificationQuery strQuery, [strQueryLanguage = "WQL"], [intFlags = (wbemFlagForwardOnly + wbemFlagReturnImmediately)], [objWbemNamedValueSet = null]	Executes a query to receive events.
	ExecNotificationQueryAsync objWbemSink, strQuery, [strQueryLanguage = "WQL"], [intFlags = 0], [objwbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously executes a query to receive events.
	ExecQuery strQuery, [strQueryLanguage = "WQL"], [intFlags = wbemFlagReturnImmediately], [objWbemNamedValueSet = null]	Executes a query to retrieve a collection of objects (classes or instances).

Table 4.8 *The SWbemServices Object (continued)*

SWbemServices (with SWbemServicesEx)		
Methods (cont'd.)	ExecQueryAsync [objWbemSink], strQuery, [strQueryLanguage = "WQL"], [intFlags], [objwbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously executes a query to retrieve a collection of objects (classes or instances).
	Get [strObjectPath = ""], [intFlags = 0], [objWbemNamedValueSet = null]	Retrieves a class or instance.
	GetAsync objWbemSink = null, [strObjectPath = ""], [intFlags = 0], [objwbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously retrieves a class or instance.
	InstancesOf strClass, [intFlags = wbemFlagReturnImmediately], [objWbemNamedValueSet = null]	Returns a collection of instances of a specified class.
	InstancesOfAsync objWbemSink, strClass, [intFlags], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously returns a collection of instances of a specified class.
	Put objWbemObject, [intFlags = 0], [objWbemNamedValueSet = null]	Saves the object to the namespace bound to the SWbemServicesEx object.
	PutAsync objWbemSink, ojbWbemObject, [intFlags = 0], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Saves the object to the namespace bound to the SWbemServicesEx object asynchronously.

Table 4.8 *The SWbemServices Object (continued)*

SWbemServices (with SWbemServicesEx)		
Methods (cont'd.)	ReferencesTo strObjectPath, [strResultClass = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredQualifier = ""], [intFlags = wbemFlagReturnImmediately], [objWbemNamedValueSet = null]	Returns a collection of objects (classes or instances) that refer to a single object. This method performs the same function that the REFERENCES OF WQL query performs.
	ReferencesToAsync objWbemSink, strObjectPath, [strResultClass = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredQualifier = ""], [intFlags], [objWbemNamedValueSet = null], [objWbemAsyncContext = null]	Asynchronously returns a collection of objects (classes or instances) that refer to a single object. This method performs the same function that the REFERENCES OF WQL query performs.
	SubclassesOf [strSuperclass = ""], [intFlags = wbemFlagReturnImmediately+wbemQueryFlagDeep], [objWbemNamedValueSet = null]	Returns a collection of subclasses of a specified class.
	SubclassesOfAsync objWbemSink, [strSuperclass = ""], [intFlags = wbemQueryFlagDeep], [objwbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously returns a collection of subclasses of a specified class.

methods, keep in mind that the **SWbemServices** object creation can be achieved with the moniker and that you can combine any of the methods exposed by the **SWbemServices** object.

The same rule applies for the **SWbemObject** object. This may seem very detailed, but sometimes it can be easier to code such a combination than to use the explicit WMI scripting API objects to create the **SWbemServices** object, as in Sample 4.3 ("Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege strings set at the SWbemLocator object level") and Sample 4.5 ("Obtaining an SWbemServices object with the WMI scripting API and the WMI privilege strings set at the SWbemServices object level"). From this point of view, WMI scripting is very versatile and offers many ways to code a script to access information.

If instead of using the line:

```
Set objWMIInstance = objWMIServices.Get ("Win32_Service='SNMP'")
```

the script uses the line:

```
Set objWMIInstances = objWMIServices.InstancesOf ("Win32_Service")
```

Then, the script retrieves a collection of instances from the *Win32_Service* class represented by the **SWbemObjectSet** (number 14 in Figure 4.3). So, instead of retrieving one instance as the SNMP service (an **SWbemObject** located by number 19 in Figure 4.3), the *InstancesOf* method returns a list of all *Win32_Service* instances available in a Windows system. Sample 4.1 ("A VBScript listing the Win32_Service instance properties"), shown at the beginning of this chapter, lists the properties of the SNMP service. Now, because the returned object is a collection, a small adaptation with a For Each loop lists the properties of all Windows services. Sample 4.6 shows the script logic.

Sample 4.6 *Retrieving all instances of the Win32_Service class with their properties*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
..:
27:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
```

```
28:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
29:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
30:    Set objWMIInstances = objWMIServices.InstancesOf (cWMIClass)
31:
32:    For Each objWMIInstance in objWMIInstances
33:        WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
34:        WScript.Echo "   AcceptPause=" & objWMIInstance.AcceptPause
35:        WScript.Echo "   AcceptStop=" & objWMIInstance.AcceptStop
36:        WScript.Echo "   Caption=" & objWMIInstance.Caption
..:
54:        WScript.Echo "   Status=" & objWMIInstance.Status
..:
59:    Next
..:
64:    ]]>
65:    </script>
66:  </job>
67:</package>
```

The *InstancesOf* method retrieves the collection. Note that using another **SWbemServices** method with a WQL query produces the same effect. So, line 30 can be changed to

```
Set objWMIInstances = objWMIServices.ExecQuery ("Select * From " & cWMIClass)
```

Previously, the script retrieved an instance of the SNMP service with the *Get* method of the **SWbemServices** object. The line using the *Get* method can also be replaced by a query returning one instance. So, the query becomes

```
Set objWMIInstances = objWMIServices.ExecQuery ("Select * From Win32_Service Where Name='SNMP'")
```

Even if there is only one instance of the SNMP service in the system, the query always returns an **SWbemObject** object collection. As usual, this collection is represented by an **SWbemObjectSet** object (number 14 in Figure 4.3). In this case, to display the properties of the SNMP service, the same tactic as used in Sample 4.6 with the For Each loop must be used. The **SWbemObjectSet** properties and methods are shown in Table 4.9. Notice that the **SWbemObjectSet** also has a security property to set the security

Table 4.9 *The SWbemObjectSet Object*

SWbemObjectSet		
Properties	Count	The number of items in the collection.
	Security	Used to read or change the security settings.
Methods	Item strObjectPath	Retrieves an SWbemObject object from the collection. This is the default method of the object.

settings. This possibility is shown in the WMI object model represented in Figure 4.3 by numbers 15, 16, and 17.

In addition to the property values retrieval of the *Win32_Service* real-world manageable object (represented by the **SWbemObject**), Table 4.1 also shows some methods associated with the *Win32_Service* class.

For an example of how to use one of the methods available from the *Win32_Service* class, imagine a script stopping a service when the service is running and starting the service when the service is stopped. This logic is implemented in Sample 4.7.

Sample 4.7 *Using one method of a Win32_Service instance directly*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
29:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
30:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
31:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
32:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
33:
34:    WScript.Echo vbCRLF & "'" & objWMIInstance.DisplayName & "' is currently " & _
35:                 objWMIInstance.State & "."
36:
37:    If objWMIInstance.State = "Stopped" Then
38:       intRC = objWMIInstance.StartService
39:       strMessage = "'" & objWMIInstance.DisplayName & "' is now Running."
40:    End If
41:    If objWMIInstance.State = "Running" Then
42:       intRC = objWMIInstance.StopService
43:       strMessage = "'" & objWMIInstance.DisplayName & "' is now Stopped ."
44:    End If
45:
46:    If intRC = 0 Then
47:       Wscript.Echo strMessage
48:    Else
49:       Wscript.Echo "Synchronous method execution failed."
50:    End If
..:
55:    ]]>
56:    </script>
57:  </job>
58:</package>
```

At lines 37 and 41, the script tests the status of the service. If the service is stopped, the script starts the service (line 38) and prepares a corresponding message (line 39). If the service is started, the script stops the service (line 42) and prepares the corresponding message (line 43). In both cases, the script tests the return code of the method execution and displays the output message accordingly (lines 46 to 50). As the method name is directly coded in the script code (lines 38 and 42), this technique refers to the direct method execution. In Section 4.4.4.4, we see another way to execute methods called indirect method execution.

4.4.3 Retrieving WMI class relationships

In Chapter 2 we saw that a class can be associated to other classes. In this case, CIM talks about association and implies the creation of references. We also saw that a class can be derived to create one or more subclasses. In such a case, there is an inheritance between classes. With the **SWbemServices** object methods, it is possible to retrieve the relations that a class has with other classes.

4.4.3.1 The subclasses

To retrieve the subclasses of a given class, the line invoking the **SWbemServices** method must be changed to

```
Set objWMIInstances = objWMIServices.SubClassesOf ("CIM_Service", wbemQueryFlagShallow)
```

The *CIM_Service* class is used here because the *Win32_Service* has no subclasses. In fact, the *Win32_Service* is not a direct subclass of the *CIM_Service*. *Win32_Service* is a direct subclass of *Win32_BaseService* class, and *Win32_BaseService* is a direct subclass of *CIM_Service* class. You can use the **WMI CIM Studio** tool to see this class implementation.

By default, the *SubClassesOf* method returns all the classes that are subclasses of the base class given, regardless of whether subclasses inherit directly or indirectly from the base class. To retrieve the list of classes that inherit directly from the base class only, a constant must be provided (**wbemQueryFlagShallow**) to modify the default behavior of the *SubClassesOf* method. Because the inheritance relationship between the classes in the CIM repository is arborescent, it is possible to retrieve the tree of the classes by using a recursive algorithm, as shown in Sample 4.8.

Sample 4.8 *Retrieving the class inheritance tree with the SWbemServices object*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "CIM_ManagedSystemElement"
..:
27:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
28:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
29:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
30:
31:    DisplaySubClasses objWMIServices, cWMIClass
..:
35:      ------------------------------------------------------------------------------
36:    Function DisplaySubClasses (ByVal objWMIServices, ByVal strWMIClass)
..:
42:        Set objWMIInstances = objWMIServices.SubClassesOf (strWMIClass, wbemQueryFlagShallow)
43:
44:        WScript.Echo Space (intIndice) & strWMIClass
45:
46:        For Each objWMIInstance in objWMIInstances
47:            DisplaySubClasses objWMIServices, objWMIInstance.Path_.RelPath
48:        Next
..:
54:    End Function
55:
56:    ]]>
57:    </script>
58:  </job>
59:</package>
```

Because the script retrieves the inheritances, the code starts from the higher class available in the CIM Core Schema: *CIM_ManagedSystemElement* (see Chapter 2) defined in line 22. The script creates the **SWbemServices** object (lines 27 to 29) and calls the DisplaySubClasses() function used recursively (lines 31 and 47). In this function, the script invokes the *SubClassesOf* method exposed by the **SWbemServices** object. This retrieves an **SWbemObject** collection contained in an **SWbemObjectSet** (line 42). If the class has some subclasses, the collection is enumerated with the For Each loop (lines 46 to 48). For each item in the collection, the function calls itself to display the related subclasses (line 44 and 47). The function that retrieves the name of the subclass uses two objects in a single line (line 47). Because each item in

the collection represents an instance of the class in the **SWbemObject** object, the **SWbemObject** object exposes a property called *Path_*. This property produces a new object called **SWbemObjectPath**. This last object exposes another property called *Relpath* containing the name of the class. Later, we examine the properties and methods exposed by the **SWbemObject**. Running the script produces the following output:

```
CIM_ManagedSystemElement
  CIM_LogicalElement
   CIM_SystemResource
    CIM_IRQ
     Win32_IRQResource
    CIM_MemoryMappedIO
     Win32_SystemMemoryResource
      Win32_DeviceMemoryAddress
      Win32_PortResource
    CIM_DMA
     Win32_DMAChannel
    Win32_Environment
   CIM_System
    CIM_ComputerSystem
     CIM_UnitaryComputerSystem
      Win32_ComputerSystem
    CIM_ApplicationSystem
   CIM_Service
    CIM_ClusteringService
    CIM_BootService
    Win32_BaseService
     Win32_SystemDriver
     Win32_Service
    Win32_ApplicationService
    ...
    ...
    ...
```

This script will be the basic foundation of a script browsing the CIM repository.

4.4.3.2 The associations and the references

To retrieve the associations and the references related to the *Win32_Service*, the *AssociatorsOf* and *ReferencesTo* methods must be used. Sample 4.9 retrieves the associations from the *Win32_Service* class.

Sample 4.9 *Retrieving the class associations from the SWbemServices object for a CIM class*

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

```
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
..:
27:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
28:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
29:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
30:    Set objWMIInstances = objWMIServices.AssociatorsOf (cWMIClass,,,,,,True)
31:
32:    For Each objWMIInstance in objWMIInstances
33:        WScript.Echo objWMIInstance.Path_ & " (" & objWMIInstance.Path_.RelPath & ")"
34:    Next
35:
36:    ]]>
37:    </script>
38:  </job>
39:</package>
```

By default, the *AssociatorsOf* method expects to have an object instance and not a class instance. Because the purpose of the script is to explore the CIM schema, a specific parameter must be given. This parameter corresponds to the True value given in the parameters of the *AssociatorsOf* method (line 30). This forces the examination of the classes instead of the instances of the classes. As a result, the CIM Schema is examined and produces the following output:

```
C:\>AssociatorsOfObjectClassWithAPI.wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

\\NET-DPEN6400A\ROOT\CIMV2:Win32_WMISetting (Win32_WMISetting)
\\NET-DPEN6400A\ROOT\CIMV2:Win32_ServiceSpecification (Win32_ServiceSpecification)
\\NET-DPEN6400A\ROOT\CIMV2:Win32_ComputerSystem (Win32_ComputerSystem)
```

As we can see, three classes are associated with the *Win32_Service* class: the *Win32_WMISetting*, the *Win32_ServiceSpecification*, and the *Win32_ComputerSystem*. Modifying line 30 of the previous code to use the *ReferencesTo* method to retrieve the references included in the *Win32_Service* class produces the code shown in Sample 4.10.

Sample 4.10 *Retrieving the class references from the SWbemServices object for a CIM class*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
```

```
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
..:
27:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
28:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
29:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
30:    Set objWMIInstances = objWMIServices.ReferencesTo (cWMIClass,,,,True)
31:
32:    For Each objWMIInstance in objWMIInstances
33:        WScript.Echo objWMIInstance.Path_ & " (" & objWMIInstance.Path_.RelPath & ")"
34:    Next
35:
36:    ]]>
37:    </script>
38:  </job>
39:</package>
```

In this case the output will be

```
C:\>ReferencesOfObjectClassWithAPI.wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

\\NET-DPEN6400A\ROOT\CIMV2:Win32_WMIElementSetting (Win32_WMIElementSetting)
\\NET-DPEN6400A\ROOT\CIMV2:Win32_ServiceSpecificationService (Win32_ServiceSpecificationService)
\\NET-DPEN6400A\ROOT\CIMV2:Win32_SystemServices (Win32_SystemServices)
```

If you check the output for Sample 4.9 ("Retrieving the class associations from the SWbemServices object for a CIM class") and Sample 4.10 ("Retrieving the class references from the SWbemServices object for a CIM class") with the **WMI CIM Studio** tool, you will see that you retrieve the same information type when looking at the association pane of the *Win32_Service* class. The scripts retrieve the same associations and references defined with the *Win32_Service* class as seen in Section 2.9.1. If you remember, in that section, we used **WMI CIM Studio** to examine the dependencies of the MSExchangeSA service. If we modify Sample 4.9 to retrieve the associations of the *Win32_Service* MSExchangeSA instance, we obtain the MSExchangeSA Windows service dependencies. So, line 30

```
Set objWMIInstances = objWMIServices.AssociatorsOf (cWMIClass,,,,,,True)
```

becomes

```
Set objWMIInstances = objWMIServices.AssociatorsOf ("Win32_Service='MSExchangeSA'")
```

Sample 4.11 shows the same script as Sample 4.9 with different parameters for the *AssociatorsOf* method.

Sample 4.11 *Retrieving the class Associations from the SWbemServices object for a CIM class instance*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "MSExchangeSA"
..:
28:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
29:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
30:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
31:    Set objWMIInstances = objWMIServices.AssociatorsOf (cWMIClass & "='" & cWMIInstance & "'")
32:
33:    For Each objWMIInstance in objWMIInstances
34:        WScript.Echo objWMIInstance.Path_ & " (" & objWMIInstance.Path_.RelPath & ")"
35:    Next
..:
40:    ]]>
41:    </script>
42:  </job>
43:</package>
```

In this case, the output will be

```
C:\>AssociatorsOfObjectInstanceWithAPI.wsfà
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

\\NET-DPEN6400A\root\cimv2:Win32_ComputerSystem.Name="NET-DPEN6400A"
Win32_ComputerSystem.Name="NET-DPEN6400A")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="MSExchangeIS" (Win32_Service.Name="MSExchangeIS")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="MSExchangeMTA" (Win32_Service.Name="MSExchangeMTA")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="Eventlog" (Win32_Service.Name="Eventlog")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="NtLmSsp" (Win32_Service.Name="NtLmSsp")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="RpcLocator" (Win32_Service.Name="RpcLocator")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="RpcSs" (Win32_Service.Name="RpcSs")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="lanmanworkstation"
(Win32_Service.Name="lanmanworkstation")
\\NET-DPEN6400A\root\cimv2:Win32_Service.Name="lanmanserver" (Win32_Service.Name="lanmanserver")
```

By referring to Section 2.9.1 or by using the **WMI CIM Studio**, you will see that the dependency list is exactly the same as the output produced by the script. Now, to determine the direction of the dependency, the script must use the *ReferencesOf* method. If we proceed in the same way, we

replace line 30 of Sample 4.10 ("Retrieving the class references from the SWbemServices object for a CIM class") with

```
Set objWMIInstances = objWMIServices.ReferencesTo ("Win32_Service='MSExchangeSA'")
```

In this case, the output will be

```
C:\>ReferencesOfObjectInstanceWithQuery.wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

\\NET-DPEN6400A\root\cimv2:Win32_SystemServices
.GroupComponent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_ComputerSystem.Name=\"NET-DPEN6400A\""
,PartComponent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
(Win32_SystemServices
.GroupComponent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_ComputerSystem.Name=\"NET-DPEN6400A\""
,PartComponent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeMTA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeMTA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"Eventlog\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"Eventlog\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"NtLmSsp\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"NtLmSsp\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"RpcLocator\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"RpcLocator\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"RpcSs\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"RpcSs\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")

\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"lanmanworkstation\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"lanmanworkstation\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")
```

```
\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"lanmanserver\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"lanmanserver\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\"")
```

To improve readability, a blank line separates each reference. So, for one *Win32_Service* reference, we have

```
\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\"" (Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\"")
```

In Section 2.9.1, we saw that the *Win32_DependentService* association class has two defined keys: *Antecedent* and *Dependent* (shown in bold in the sample output line). If we decompose this line, we have

```
\\NET-DPEN6400A\root\cimv2:Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\""

(Win32_DependentService
.Antecedent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeSA\""
,Dependent="\\\\NET-DPEN6400A\\root\\cimv2:Win32_Service.Name=\"MSExchangeIS\"")
```

where the name of the association class instance is composed by the name of the keys defined in the *Win32_DependentService* class.

As we saw when learning WQL (Chapter 3), it is possible to retrieve the list of associations and references by using a WQL query. In this case, the line for the associations will be

```
Set objWMIInstances = objWMIServices.ExecQuery ("ASSOCIATORS OF {Win32_Service='MSExchangeSA'}")
```

For the references, the line will be

```
Set objWMIInstances = objWMIServices.ExecQuery ("REFERENCES OF {Win32_Service='MSExchangeSA'}")
```

The output is exactly the same. It is just another way to get the same information.

4.4.4 Retrieving the WMI object CIM information

Besides the relationships that a class definition may have in the CIM repository, there is another set of information that is important to know when writing scripts with the CIM classes. This information concerns the following:

- The nature of the class
- The properties exposed by the class

- The nature of the properties exposed by the class
- The methods exposed by the class
- The nature of the methods exposed by the class.
- The method properties exposed by the class
- The nature of the method properties exposed by the class

The following sections continue to explore the WMI scripting API object model and explain how this information can be retrieved by script from the CIM repository.

4.4.4.1 The CIM class qualifiers

As we saw in Chapter 2, a class can have various forms, such as static, dynamic, system, abstract, and association. This is determined with a qualifier. Every element defined in the CIM repository has qualifiers determining its nature and its behavior. To retrieve the class qualifier, we must examine the properties and methods exposed by the **SWbemObject** object in Table 4.10.

The portion of the WMI object model that represents **SWbemObject** is shown in Figure 4.4.

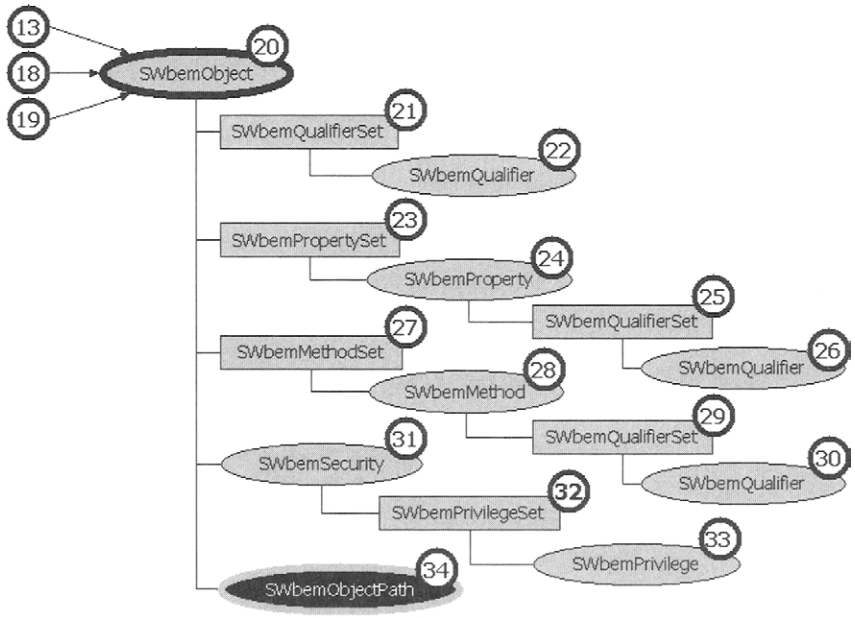


Figure 4.4
The WMI object model (2/3).

Table 4.10 *The SWbemObject Object*

SWbemObject (with SWbemObjectEx)		
Properties	Derivation_	Contains an array of strings describing the derivation hierarchy for the class.
	Methods_	An SWbemMethodSet object that is the collection of methods for this object.
	Path_	Contains an SWbemObjectPath object that represents the object path of the current class or instance.
	Properties_	An SWbemPropertySet object that is the collection of properties for this object.
	Qualifiers_	An SWbemQualifierSet object that is the collection of qualifiers for this object.
	Security_	Contains an SWbemSecurity object used to read or change the security settings.
	SystemProperties_	An SWbemPropertySet object containing the collection of system properties that apply to the SWbemObjectEx represents.
Methods	Associators_ [strAssocClass = ""], [strResultClass = ""], [strResultRole = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredAssocQualifier = ""], [strRequiredQualifier = ""], [intFlags = wbemFlagReturnImmediately], [objwbemNamedValueSet = null]	Retrieves the associators of the object.

Table 4.10 *The SWbemObject Object (continued)*

SWbemObject (with SWbemObjectEx)		
Methods (cont'd.)	AssociatorsAsync_ objWbemSink, [strAssocClass = ""], [strResultClass = ""], [strResultRole = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredAssocQualifier = ""], [strRequiredQualifier = ""], [intFlags], [objwbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously retrieves the associators of the object.
	Clone_ ()	Makes a copy of the current object.
	CompareTo_ objwbemObject, [intFlags = wbemComparisonFlagIncludeAll]	Tests two objects for equality.
	Delete_ [intFlags = 0], [objwbemNamedValueSet = null]	Deletes the object from WMI.
	DeleteAsync_ objWbemSink, [intFlags = 0], [objwbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously deletes the object from WMI.
	ExecMethod_ strMethodName, [objwbemInParams = null], [intFlags = 0], [objwbemNamedValueSet = null]	Executes a method exported by a method provider.

Table 4.10 *The SWbemObject Object (continued)*

SWbemObject (with SWbemObjectEx)		
Methods (cont'd.)	ExecMethodAsync_ objWbemSink, strMethodName, [objwbemInParams = null], [intFlags = 0], [objwbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously executes a method exported by a method provider.
	GetObjectText_ [intFlags = 0]	Retrieves the textual representation of the object (MOF syntax).
	Instances_ [intFlags = wbemFlagReturnImmediately], [objwbemNamedValueSet = null]	Returns a collection of instances of the object (which must be a WMI class).
	InstancesAsync_ [objWbemSink], [intFlags], [objwbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously returns a collection of instances of the object (which must be a WMI class).
	Put_ [intFlags = wbemChangeFlagCreateOrUpdate], [objwbemNamedValueSet = null]	Creates or updates the object in WMI.
	PutAsync_ objWbemSink, [intFlags = wbemChangeFlagCreateOrUpdate], [objWbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously creates or updates the object in WMI.
	References_ [strResultClass = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredQualifier = ""], [intFlags = wbemFlagReturnImmediately], [objwbemNamedValueSet = null]	Returns references to the object.

Table 4.10 *The SWbemObject Object (continued)*

SWbemObject (with SWbemObjectEx)		
Methods (cont'd.)	ReferencesAsync_ objWbemSink, [strResultClass = ""], [strRole = ""], [boolClassesOnly = FALSE], [boolSchemaOnly = FALSE], [strRequiredQualifier = ""], [intFlags], [objwbemNamedValueSet = null], S[objWbemAsyncContext]	Asynchronously returns references to the object.
	SpawnDerivedClass_ [intFlags = 0]	Creates a new derived class from the current object (which must be a WMI class).
	SpawnInstance_ [intFlags = 0]	Creates a new instance from the current object.
	Subclasses_ [intFlags = wbemFlagReturnImmediately+wbemQueryFlagDeep], [objwbemNamedValueSet = null]	Returns a collection of subclasses of the object (which must be a WMI class).
	SubclassesAsync_ objWbemSink, [intFlags = wbemQueryFlagDeep], [objWbemNamedValueSet = null], [objWbemAsyncContext]	Asynchronously returns a collection of subclasses of the object (which must be a WMI class).
	GetText_ intTextFormat, [intFlags = 0], [objWbemNamedValueSet]	Returns a text file showing the contents of an object in XML.
	Refresh_ [intFlags = 0], [objWbemNamedValueSet = null]	Refreshes data in an object.

Table 4.11 *The SWbemQualifierSet Object*

SWbemQualifierSet		
Properties	Count	Number of items in the collection.
Methods	Add strName, varVal, [boolPropagatesToSubclasses = TRUE], [boolPropagatesToInstances = TRUE], [boolOverridable = TRUE], [intFlags = 0]	
	Item strName, [intFlags = 0]	Adds an SWbemQualifier object to the SWbemQualifierSet collection.
	Remove strName, [intFlags = 0]	Retrieves an SWbemQualifier object from the collection. This is the default method of this object.
	Add strName, varVal, [boolPropagatesToSubclasses = TRUE], [boolPropagatesToInstances = TRUE], [boolOverridable = TRUE], [intFlags = 0]	Removes an SWbemQualifier object from the collection.

The **SWbemObject** object exposes a *Qualifier_* property to retrieve an **SWbemQualifierSet** object (located as number 21 in Figure 4.4). As we saw in Chapter 2, we may have several qualifiers for a CIM element. The list of qualifiers associated with a CIM element is retrieved in the form of a collection and can be enumerated to retrieve every item part of the collection. Table 4.11 shows the list of properties and methods exposed by the **SWbemQualifierSet** object.

Each item in the **SWbemQualifierSet** object will be an **SWbemQualifier** object (located in Figure 4.4 as number 22). Table 4.12 shows the list of properties exposed by the **SWbemQualifier** object.

Table 4.12 *The SWbemQualifier Object*

SWbemQualifier		
Properties	IsAmended	Boolean value that indicates if this qualifier has been localized using a merge operation
	IsLocal	Boolean value that indicates if this is a local qualifier.
	IsOverridable	Boolean value that indicates if this qualifier can be overridden when propagated.

Table 4.12 *The SWbemQualifier Object (continued)*

SWbemQualifier		
Properties (cont'd.)	Name	Name of this qualifier.
	PropagatesToInstance	Boolean value that indicates if this qualifier can be propagated to an instance.
	PropagatesToSubclass	Boolean value that indicates if this qualifier can be propagated to a subclass.
	Value	Actual value of this qualifier. This is the default property of this object.

If we use a piece of sample script to retrieve an instance of the *Win32_Service* class as defined in the CIM repository and make use of the *Qualifier_* property associated with the retrieved **SWbemObject**, as shown in Sample 4.12, it is possible to retrieve the list of the class qualifiers with their properties.

Sample 4.12 *Retrieving the class qualifiers from the SWbemObject object for a CIM class*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
..:
29:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
30:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
31:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
32:
33:    Set objWMIInstance = objWMIServices.Get (cWMIClass, wbemFlagUseAmendedQualifiers)
34:
35:    Wscript.Echo String (60, 45)
36:    Wscript.Echo "Class Qualifiers:"
37:
38:    WScript.Echo "  " & objWMIInstance.Path_.RelPath
39:
40:    intIndent = 4
41:
42:    Set objWMIQualifiers = objWMIInstance.Qualifiers_
43:    For Each objWMIQualifier In objWMIQualifiers
44:        WScript.Echo Space (intIndent) & "Name: " & objWMIQualifier.Name
45:        If IsArray (objWMIQualifier.Value) Then
46:           For Each varElement In objWMIQualifier.Value
```

```
47:              WScript.Echo Space (intIndent) & _
48:                           "  Value:                    " & varElement
49:           Next
50:        Else
51:           WScript.Echo Space (intIndent) & _
52:                        "  Value:                  " & objWMIQualifier.Value
53:        End if
54:        WScript.Echo Space (intIndent) & _
55:                     "  Amended:                " & objWMIQualifier.IsAmended
56:        WScript.Echo Space (intIndent) & _
57:                     "  Local:                  " & objWMIQualifier.IsLocal
58:        WScript.Echo Space (intIndent) & _
59:                     "  Overridable:            " & objWMIQualifier.IsOverridable
60:        WScript.Echo Space (intIndent) & _
61:                     "  Propagates to instance: " & objWMIQualifier.PropagatesToInstance
62:        WScript.Echo Space (intIndent) & _
63:                     "  Propagates to subclass: " & objWMIQualifier.PropagatesToSubclass
64:    Next
 ..:
69:    ]]>
70:    </script>
71:  </job>
72:</package>
```

The script structure from line 20 to line 33 is the same as before. However, to retrieve the instance of the CIM class in the **SWbemObject** (line 33), a particular flag is used with *Get* method. Because qualifiers are localized (or amended), if we want to retrieve all qualifiers available, it is important to specify the wbemFlagUseAmendedQualifiers constant to ensure that the script retrieves the localized data. This is an important parameter because, for example, the *Description* qualifier is a localized qualifier and contains a descriptive text about the CIM element (class, property, method). As mentioned before, when opening a language-specific class, WMI combines the language-neutral definition with the language-specific definition to provide the localized version of the class. The wbemFlagUseAmendedQualifiers constant is part of the wbemFlagEnum constants enumeration (Table 4.13). The constants of this collection can be used from the *ExecQuery* method, the *ExecQueryAsync* method, the *SubclassesOf* method, and the *InstancesOf* method of the **SWbemServices** object.

Table 4.13 *The wbemFlagEnum Constants*

WbemFlagEnum		
Properties	*Value*	*Description*
wbemFlagReturnImmediately	16	Causes the call to return immediately.
wbemFlagReturnWhenComplete	0	Causes this call to block until the call has completed.
wbemFlagBidirectional	0	Causes WMI to retain pointers to objects of the enumeration until the client releases the enumerator.

Table 4.13 *The wbemFlagEnum Constants (continued)*

WbemFlagEnum		
Properties	*Value*	*Description*
wbemFlagForwardOnly	32	Causes a forward-only enumerator to be returned. Use this flag in combination with wbemFlagReturnImmediately to request semi-synchronous access. For more information, see Making a Semi-synchronous Call. Forward-only enumerators are generally much faster and use less memory than conventional enumerators, but they do not allow calls to Clone or Reset.
wbemFlagNoErrorObject	64	Causes asynchronous calls to not return an error object in the event of an error.
wbemFlagReturnErrorObject	0	Causes asynchronous calls to return an error object in the event of an error.
wbemFlagSendStatus	128	Causes asynchronous calls to send status updates to the **OnProgress** event handler for your object sink.
wbemFlagDontSendStatus	0	Prevents asynchronous calls from sending status updates to the **OnProgress** event handler for your object sink.
wbemFlagUseAmendedQualifiers	131072	Causes asynchronous calls to send status updates to the OnProgress event handler for your object sink.

Next, the *Qualifier_* property is used to get an **SWbemQualifierSet** object (Sample 4.12, line 42). A For Each loop is executed to enumerate every qualifier contained in the collection (lines 43 to 64). In the loop, every property exposed by the **SWbemQualifier** object (see Table 4.12) is displayed. Special care is taken to make sure that the value contained in the qualifier is not an array (lines 45). In such a case, the value is processed accordingly (lines 46 to 49). For the *Win32_Service* class, the script retrieves five qualifiers (description, dynamic, locale, provider and UUID) displayed as follows:

```
1:C:\>QualifierForObjectClassWithAPI.wsf
2:Microsoft (R) Windows Script Host Version 5.6
3:Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:-----------------------------------------------------------
6:Class Qualifiers:
7:  Win32_Service
8:     Name: Description
9:       Value:                  The Win32_Service class represents a service on a Win32 computer
                                 system. A service application conforms to the interface rules of
                                 the Service Control Manager (SCM) and can be started by a user
                                 automatically at system boot through the Services control panel
                                 utility, or by an application that uses the service functions
                                 included in the Win32 API. Services can execute even when no user
                                 is logged on to the system.
```

```
10:     Amended:                 True
11:     Local:                   True
12:     Overridable:             True
13:     Propagates to instance: False
14:     Propagates to subclass: True
15:   Name: dynamic
16:     Value:                   True
17:     Amended:                 False
18:     Local:                   True
19:     Overridable:             True
20:     Propagates to instance: True
21:     Propagates to subclass: False
22:   Name: Locale
23:     Value:                   1033
24:     Amended:                 True
25:     Local:                   True
26:     Overridable:             True
27:     Propagates to instance: False
28:     Propagates to subclass: False
29:   Name: provider
30:     Value:                   CIMWin32
31:     Amended:                 False
32:     Local:                   True
33:     Overridable:             True
34:     Propagates to instance: True
35:     Propagates to subclass: False
36:   Name: UUID
37:     Value:                   {8502C4D9-5FBB-11D2-AAC1-006008C78BC7}
38:     Amended:                 False
39:     Local:                   True
40:     Overridable:             True
41:     Propagates to instance: True
42:     Propagates to subclass: False
```

For instance, this information tells us that the *Win32_Service* class is a dynamic class (lines 15 and 16) implemented by the *CIMWin32* provider (lines 29 and 30). We know that the class contains localized information (lines 11, 18, 25, 32, and 39). We can also see that every qualifier is differently propagated to instances and subclasses (lines 13 and 14, 20 and 21, 27 and 28, 34 and 35, 41 and 42).

4.4.4.2 The CIM class properties

Because we can retrieve information about the class qualifiers, we can also retrieve the list of properties exposed by a WMI class. In previous samples, it was necessary to display a property of the *Win32_Service* instance; the code contained the property in line. For instance, to get the name of the service instance retrieved, as in Sample 4.1 ("A VBScript listing the Win32_Service instance properties") and Sample 4.6 ("Retrieving all instances of the Win32_Service class with their properties"), the line was as follows:

```
WScript.Echo "   DisplayName=" & objWMIInstance.Name
```

Table 4.14 *The SWbemProperty Object*

Properties	CIMType	Type of this property.
	IsArray	Boolean value that indicates if this property has an array type.
	IsLocal	Boolean value that indicates if this is a local property.
	Name	Name of this property.
	Origin	Contains the originating class of this property.
	Qualifiers_	An SWbemQualifierSet object, which is the collection of qualifiers for this property.
	Value	Actual value of this property. This is the default automation property of this object.

This coding technique is referred to as *direct property access*. This way of coding is perfectly fine, given the condition that the **SWbemObject** object retrieved has a property called *name*. While the code considers a class exposing this property, there is no problem. If the class is different, it is likely that some properties are different, and the script code must be modified accordingly. It depends on the context of the developed script, but it could be interesting to develop code that is generic enough to retrieve the property list from an object class and for each property found to retrieve the associated value. This way of working is referred to as *indirect property access*.

The **SWbemObject** object exposes a property called *Properties_* (see Table 4.10), which produces an **SWbemPropertySet** object (located as number 23 in Figure 4.4). This object represents a collection of properties supported by the **SWbemObject** object. Each property is contained in an **SWbemProperty** object (located as number 24 in Figure 4.4). The **SWbemPropertySet** and the **SWbemProperty** objects properties and methods are available in Tables 4.14 and 4.15.

Table 4.15 *The SWbemPropertySet Object*

Properties	Count	The number of items in the collection.
Methods	Add strName, intCIMType, [boolIsArray = FALSE], [intFlags = 0]	Adds an SWbemProperty object to the SWbemPropertySet collection.
	Item strName, [intFlags = 0]	Retrieves an SWbemProperty from the collection. This is the default method of this object.
	Remove strName, [iFlags = 0]	Removes an SWbemProperty object from the collection.

Sample 4.13 illustrates the properties retrieval technique. The basic structure of the script is the same as prior scripts, except that the way to retrieve the *Win32_Service* properties differs. Note that the script retrieves an instance of the *Win32_Service* class (line 33), which makes it possible for the script to display the values of every retrieved property from that instance.

Sample 4.13 *Retrieving the property collection of a WMI instance*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
34:
35:    WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
36:
37:    Set objWMIPropertySet = objWMIInstance.Properties_
38:    For Each objWMIProperty In objWMIPropertySet
39:        If objWMIProperty.IsArray Then
40:           For Each varElement In objWMIProperty.Value
41:               WScript.Echo "  " & objWMIProperty.Name & " (" & varElement & ")"
42:           Next
43:        Else
44:           WScript.Echo "  " & objWMIProperty.Name & " (" & objWMIProperty.Value & ")"
45:        End If
46:    Next
..:
52:    ]]>
53:    </script>
54:  </job>
55:</package>
```

At line 37, the script refers to the **SWbemObject** *Properties_* property to get the property collection in an **SWbemPropertySet** object. Since this object represents a property collection, the For Each loop (lines 38 to 46) examines each property. To display the value of the property, the script refers to the *IsArray* property of the **SWbemProperty** object (line 39). If it is an array, the script makes another For Each loop to display the array values.

In both cases, the script shows the name of the property and its associated value or values (lines 41 and 44).

The *Win32_Service* has properties that are read only! There is no possibility for a script to change anything. Some classes also have properties that are read/write. For instance, this is the case for the *Win32_Registry* class, which exposes one read/write property called *ProposedSize*. Table 4.16 lists the properties available from the *Win32_Registry* class (there is no method exposed). Since we can read the class properties directly and indirectly, the purpose here is to show how to modify a property directly and indirectly.

The goal of the next script is to increase the registry database size limit of 10 MB. So, the script reads the current registry size and saves the new value back to the system. This logic is implemented in Sample 4.14.

Sample 4.14 *Setting one read/write property of a Win32_Registry class instance directly*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Registry"
23:    Const cWMIInstance = "Microsoft Windows 2000 Server|C:\WINNT|\Device\Harddisk0\Partition1"
..:
28:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
29:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
30:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
31:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
32:
33:    WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
34:
35:    Wscript.Echo "Current registry size is: " & objWMIInstance.ProposedSize  & " MB."
36:
37:    objWMIInstance.ProposedSize = objWMIInstance.ProposedSize + 10
38:    objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete)
39:
40:    Wscript.Echo "Current registry size is now: " & objWMIInstance.ProposedSize  & " MB."
..:
45:    ]]>
46:    </script>
47:  </job>
48:</package>
```

As before, the first part of the script is always the same. The particularity starts at line 37, where the line performs the addition of 10 MB to the

Table 4.16 *The Win32_Registry Properties*

Properties	Caption	**string** Read-only Short description (one-line string) of the object.
	CurrentSize	**uint32** Read-only Qualifiers: Units(Megabytes) Current physical size of the Win32 registry.
	Description	**string** Read-only Description of the object.
	InstallDate	**datetime** Read-only When the object was installed. A lack of a value does not indicate that the object is not installed.
	MaximumSize	**uint32** Read-only Qualifiers: Units(Megabytes) Maximum size of the Win32 registry. If the system is successful in using the ProposedSize property, MaximumSize should contain the same value.
	Name	**string** Read-only Qualifiers: Key Name of the Win32 Registry. Maximum length is 256 characters.
	ProposedSize	**uint32** Read/Write Qualifiers: Units(Megabytes) Proposed size of the Win32 registry. It is the only registry setting that can be modified, and its proposal is attempted the next time the system boots.
	Status	**string** Read-only Current status of the object. Various operational and non-operational statuses can be defined. Operational statuses include: "OK", "Degraded", and "Pred Fail" (an element, such as a SMART-enabled hard drive, may be functioning properly but predicting a failure in the near future). Non-operational statuses include: "Error", "Starting", "Stopping", and "Service". The latter, "Service", could apply during mirror-resilvering of a disk, reload of a user permissions list, or other administrative work. Not all such work is on-line, yet the managed element is neither "OK" nor in one of the other states.

current registry size. The current registry size is read directly (as the property exposing the registry size is coded in the script code) and set directly to the new value on the same line (line 37). Setting the registry size property *ProposedSize* to the new value is not enough to commit the change. For this, another **SWbemObject** object method must be used. The *Put_* method flushes the modification back to the system (line 38). Some flags are passed to this method to ensure that the instance is updated and that thefunction returns once the execution is complete, at which time the registry size modification is saved.

Sample 4.15 indirectly sets the same property of the *Win32_Registry* class.

Sample 4.15 *Setting one read/write property of a Win32_Registry class instance indirectly*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Registry"
23:    Const cWMIInstance = "Microsoft Windows 2000 Server|C:\WINNT|\Device\Harddisk0\Partition1"
..:
29:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
30:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
31:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
32:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
33:
34:    WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
35:
36:    Set objWMIPropertySet = objWMIInstance.Properties_
37:
38:    Wscript.Echo "Current registry size is: " & objWMIPropertySet.Item("ProposedSize") & " MB."
39:
40:    objWMIPropertySet.Item("ProposedSize") = objWMIPropertySet.Item("ProposedSize") + 10
41:    objWMIInstance.Put_ (wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete)
42:
43:    Wscript.Echo "Current registry size is now: " & objWMIPropertySet.Item("ProposedSize") & " MB."
..:
50:    ]]>
51:    </script>
52:  </job>
53:</package>
```

First, the script retrieves an **SWbemPropertySet** object (line 36). With this object and the *Item* method, it is possible to reference the property *Pro-*

posedSize by passing its name as a parameter of the method (line 38). The rest of the process is exactly the same as before. The current registry size is read indirectly (as the property exposing the registry size is passed as a parameter of the *Item* method) and set indirectly on the same line (line 40). The update is executed at line 41.

Clearly, the big difference between the direct method and the indirect method is that the indirect method allows the creation of a generic code because the name of the referenced property is part of a parameter. This possibility is valuable when creating a generic script to retrieve CIM class information.

4.4.4.3 The CIM class property qualifiers

The technique for retrieving the qualifiers of the properties part of a CIM class is almost the same as retrieving the qualifiers for a CIM class (see Sample 4.12 ("Retrieving the class qualifiers from the SWbemObject object for a CIM class"). The difference resides in the level of the object model where the **SWbemQualifierSet** object is retrieved. For a class, the **SWbemQualifierSet** object was retrieved from an **SWbemObject** object; for a property, the **SWbemQualifierSet** object will be retrieved from an **SWbemProperty** object. The **SWbemQualifierSet** and **SWbemQualifier** objects linked to the **SWbemProperty** object are represented in the object model in Figure 4.4 by numbers 25 and 26. The script retrieving the property qualifiers is presented in Sample 4.16.

Sample 4.16 *Retrieving the property qualifiers of a WMI class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <script language="VBScript" src="..\Functions\DisplayQualifiersFunction.vbs" />
14:
15:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:
17:    <script language="VBscript">
18:    <![CDATA[
..:
22:    Const cComputerName = "LocalHost"
23:    Const cWMINameSpace = "root/cimv2"
24:    Const cWMIClass = "Win32_Service"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:
34:    Set objWMIInstance = objWMIServices.Get (cWMIClass, wbemFlagUseAmendedQualifiers)
35:
```

```
36:    Wscript.Echo String (60, 45)
37:    Wscript.Echo "Class Qualifiers:"
38:
39:    WScript.Echo "  " & objWMIInstance.Path_.RelPath
40:    DisplayQualifiers objWMIInstance.Qualifiers_, 0
41:
42:    Wscript.Echo String (60, 45)
43:    Wscript.Echo "Property Qualifiers:"
44:
45:    Set objWMIPropertySet = objWMIInstance.Properties_
46:    For Each objWMIProperty In objWMIPropertySet
47:        WScript.Echo "  " & objWMIProperty.Name
48:        DisplayQualifiers objWMIProperty.Qualifiers_, 0
49:    Next
..:
54:    ]]>
55:    </script>
56:  </job>
57:</package>
```

The script retrieves the class qualifiers (lines 36 to 40) and the property qualifiers (lines 42 to 49). Because the tactic to display the qualifiers is the same whether it is a class qualifier or a property qualifier, the process to enumerate the qualifier properties is encapsulated in a subfunction stored in an external file (line 13). To retrieve the property qualifiers, the script must first retrieve the property list (line 45). Next, it enumerates the property list of the class (lines 46 to 49). The function displaying the qualifier properties is as follows:

```
 .:
 .:
 6:' ------------------------------------------------------------
 7:Function DisplayQualifiers (objWMIQualifiers, intIndent)
..:
12:    intIndent = intindent + 4
13:
14:    For Each objWMIQualifier In objWMIQualifiers
15:        WScript.Echo Space (intIndent) & "Name: " & objWMIQualifier.Name
16:        If IsArray (objWMIQualifier.Value) Then
17:           For Each varElement In objWMIQualifier.Value
18:               WScript.Echo Space (intIndent) & _
19:                            "  Value:                  " & varElement
20:           Next
21:        Else
22:           WScript.Echo Space (intIndent) & _
23:                        "  Value:                 " & objWMIQualifier.Value
24:        End if
25:        WScript.Echo Space (intIndent) & _
26:                     "  Amended:               " & objWMIQualifier.IsAmended
27:        WScript.Echo Space (intIndent) & _
28:                     "  Local:                 " & objWMIQualifier.IsLocal
29:        WScript.Echo Space (intIndent) & _
30:                     "  Overridable:           " & objWMIQualifier.IsOverridable
31:        WScript.Echo Space (intIndent) & _
32:                     "  Propagates to instance: " & objWMIQualifier.PropagatesToInstance
```

```
 33:        WScript.Echo Space (intIndent) & _
 34:                     "  Propagates to subclass: " & objWMIQualifier.PropagatesToSubclass
 35:    Next
 36:
 37:End Function
```

You can see that the logic is the same as in Sample 4.12 ("Retrieving the class qualifiers from the SWbemObject object for a CIM class") for the class qualifiers retrieval. The only addition concerns an input parameter to indent the qualifier output. This feature is used in subsequent samples when we examine the method qualifiers with the method parameter qualifiers. An extract of the output of the *State* property part of the *Win32_Service* class is

```
State
  Name: CIMTYPE
    Value:                  string
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: True
    Propagates to subclass: True
  Name: Description
    Value:                  The State property indicates the current state of the base service.
    Amended:                True
    Local:                  True
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: MappingStrings
    Value:                  Win32API|Service Structures|SERVICE_STATUS|dwCurrentState
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: read
    Value:                  True
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: ValueMap
    Value:                  Stopped
    Value:                  Start Pending
    Value:                  Stop Pending
    Value:                  Running
    Value:                  Continue Pending
    Value:                  Pause Pending
    Value:                  Paused
    Value:                  Unknown
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
```

Table 4.17 *The SWbemMethodSet Object*

Properties	Count	The number of items in the collection.
Methods	Item strName, [iFlags = 0]	Retrieves an SWbemMethod object from the collection. This is the default automation method of this object.

4.4.4.4 The CIM class methods

To retrieve the methods available from an **SWbemObject** object, the technique is the same as shown previously for the properties. The **SWbemObject** object exposes a property, *Methods_*, that provides an **SWbemMethodSet** object that contains the collection of methods available. Each item in the **SWbemMethodSet** object contains an **SWbemMethod** object. Both **SWbemMethodSet** and **SWbemMethod** object properties and methods are, respectively, available in Tables 4.17 and 4.18 and are located in the WMI scripting object model in Figure 4.4 as numbers 27 and 28.

By adapting Sample 4.13 ("Retrieving the property collection of a WMI instance") to suit the needs of the methods retrieval, we have Sample 4.17.

Sample 4.17 *Retrieving the method collection of a WMI class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
34:
35:    WScript.Echo objWMIInstance.Name & " (" & objWMIInstance.Description & ")"
36:
37:    Set objWMIMethodSet = objWMIInstance.Methods_
38:    For Each objWMIMethod In objWMIMethodSet
39:        WScript.Echo "  " & objWMIMethod.Name
40:    Next
..:
46:    ]]>
47:    </script>
48:  </job>
49:</package>
```

Table 4.18 *The SWbemMethod Object*

Properties	InParameters	An SWbemObject object whose properties define the input parameters for this method.
	Name	Name of the method.
	Origin	Originating class of the method.
	OutParameters	An SWbemObject object whose properties define the out parameters and return type of this method.
	Qualifiers_	An SWbemQualifierSet object that contains the qualifiers for this method.

Again, the script structure is the same as before. The only change resides in lines 37 to 40 where the **SWbemMethodSet** object is created (line 37) and enumerated as a collection to retrieve each **SWbemMethod** object (line 39) with its associated *name* property.

When talking about the method execution in Sample 4.7 ("Using one method of a Win32_Service instance directly"), the script uses a technique referred to as direct-method execution because the method name was hard coded in the script code. For instance, in the case of the *Win32_Service StartService* and *StopService* methods, the lines used were

```
intRC = objWMIInstance.StartService
```

and

```
intRC = objWMIInstance.StopService
```

The WMI scripting API allows the execution of methods indirectly. This means that it is possible to refer to an indirect way to execute a method. This technique uses the *ExecMethod* of the **SWbemServices** object (see Table 4.8) or the *ExecMethod_* of the **SWbemObject** object (see Table 4.10). The same script as Sample 4.7 can be adapted to invoke the *StartService* and *StopService* methods indirectly. Sample 4.18 shows how to proceed.

Sample 4.18 *Using one method of a Win32_Service instance indirectly*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
```

```
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
29:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
30:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
31:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
32:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
33:
34:    WScript.Echo vbCRLF & "'" & objWMIInstance.DisplayName & "' is currently " & _
35:                 objWMIInstance.State & "."
36:
37:    If objWMIInstance.State = "Stopped" Then
38:       Set objWMIOutParameters = objWMIInstance.ExecMethod_("StartService")
39:       intRC = objWMIOutParameters.ReturnValue
40:       strMessage = "'" & objWMIInstance.DisplayName & "' is now Running."
41:    End If
42:    If objWMIInstance.State = "Running" Then
43:       Set objWMIOutParameters = objWMIInstance.ExecMethod_("StopService")
44:       intRC = objWMIOutParameters.ReturnValue
45:       strMessage = "'" & objWMIInstance.DisplayName & "' is now Stopped ."
46:    End If
47:
48:    If intRC = 0 Then
49:       Wscript.Echo strMessage
50:    Else
51:       Wscript.Echo "Synchronous method execution failed."
52:    End If
..:
58:    ]]>
59:    </script>
60:  </job>
61:</package>
```

Sample 4.18 (indirect method) uses the same structure as Sample 4.7 (direct method). The particularity resides in the method invocation (lines 38 and 43). As AAWE can see, in both cases the method name is a parameter of the *ExecMethod_* method exposed by the **SWbemObject** object. The method name is no longer encoded in the script code. The second particularity concerns the returned result of the method execution. With the direct method, it is possible to retrieve the result directly in a variable; now with the indirect method execution, the return code is returned in an **SWbemObject** object (lines 38 and 43) whose properties define the output parameters and the return value of the executed method (lines 39 and 44).

The *StartService* and *StopService* methods do not require any input parameters, but how do we deal with methods invoked indirectly that require input parameters? To explain this, let's take another script sample, swapping the service startup mode. To change the startup mode of a *Win32_Service*, the *ChangeStartMode* method must be executed with one parameter. When the startup mode is automatic, the script will change it to manual and vice versa. Sample 4.19 implements this logic.

Sample 4.19 *Using one method of a Win32_Service instance indirectly with one parameter*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
31:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
32:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
33:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
34:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
35:
36:    WScript.Echo vbCRLF & "'" & objWMIInstance.DisplayName & "' startup is currently " & _
37:                 objWMIInstance.StartMode & "."
38:
39:    Set objWMIMethod = objWMIInstance.Methods_("ChangeStartMode")
40:    Set objWMIInParameters = objWMIMethod.InParameters
41:
42:    If objWMIInstance.StartMode = "Manual" Then
43:       objWMIInParameters.Properties_.Item("StartMode") = "Automatic"
44:       Set objWMIOutParameters = objWMIInstance.ExecMethod_("ChangeStartMode", _
45:                                                            objWMIInParameters)
46:       intRC = objWMIOutParameters.ReturnValue
47:       strMessage = "'" & objWMIInstance.DisplayName & "' startup mode is now Automatic."
48:    End If
49:    If objWMIInstance.StartMode = "Auto" Then
50:       objWMIInParameters.Properties_.Item("StartMode") = "Manual"
51:       Set objWMIOutParameters = objWMIInstance.ExecMethod_("ChangeStartMode", _
52:                                                            objWMIInParameters)
53:       intRC = objWMIOutParameters.ReturnValue
54:       strMessage = "'" & objWMIInstance.DisplayName & "' startup mode is now Manual."
55:    End If
56:
57:    If intRC = 0 Then
58:       Wscript.Echo strMessage
59:    Else
60:       Wscript.Echo "Synchronous method execution failed."
61:    End If
..:
69:    ]]>
70:    </script>
71:  </job>
72:</package>
```

As with previous samples, the script structure is the same. Because the invoked method has one parameter, the script must instantiate an object

used to store the method parameter. This object is an **SWbemObject** object whose properties define the input parameters of the executed method (lines 39 and 40). In the sample case, the input parameter is "StartMode" (line 43) since the examined method is *ChangeStartMode* (line 39). To set the parameter required by the method (lines 43 and 50), the **SWbemObject** object containing the input parameter refers to the *Properties_* property (see Table 4.10) to obtain an **SWbemObjectPropertySet** object (see Table 4.14), which, in turn, refers the *Item* method to set the property value. The last step executes the method (lines 44 and 51). This step is almost the same as in Sample 4.18 ("Using one method of a Win32_Service instance indirectly") except that the **SWbemObject** object containing the input parameters is passed as a parameter of the *ExecMethod_* exposed by **SWbemObject** representing the Windows service instance (lines 45 and 52).

Note that the **SWbemMethod** object created to retrieve the **SWbemObject** object containing the input parameter is created in two steps (lines 39 and 40). It is possible to create this object in one line without creating an explicit **SWbemMethod** object. In this case lines 39 and 40 can be replaced by

```
Set objWMIInParameters = objWMIInstance.Methods_("ChangeStartMode").InParameters
```

The effect is exactly the same. The reason for this initial coding is to ease the readability of the code and to show that an intermediate WMI scripting API object is used, in this case the **SWbemMethod** object.

To invoke methods, the previous scripts referred to the methods (directly or indirectly) from **SWbemObject** representing the WMI class instance. It is also possible to invoke a WMI class method from the **SWbemServices** object by invoking its *ExecMethod* method (see Table 4.8). In this case, the instance name of the real-world manageable entity is passed as a parameter. Sample 4.20 shows this technique.

Sample 4.20 *Using one method of a Win32_Service instance from the SWbemServices object*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
```

```
23:    Const cWMIInstance = "SNMP"
..:
28:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
29:
30:    Set objWMIOutParameters = objWMIServices.ExecMethod _
31:                                    (cWMIClass & "='" & cWMIInstance & "'", "StopService")
32:
33:    If objWMIOutParameters.ReturnValue = 0 Then
34:       Wscript.Echo "Synchronous method execution successful."
35:    Else
36:       Wscript.Echo "Synchronous method execution failed."
37:    End If
..:
42:    ]]>
43:    </script>
44:  </job>
45:</package>
```

As we can see in lines 30 and 31, the instance name and the method name are passed as parameters of the *ExecMethod* exposed by the **SWbemServices** object. Of course, there is no way to retrieve the status of the Windows service from an **SWbemServices** object to determine whether the script must stop or start the service. If this is required, an object representing the instance of the service must be created to read its status. Sample 4.21 performs the exact same operation, but uses a moniker and invokes the *ExecMethod* method directly from the **GetObject** statement.

Sample 4.21 *Using one method of a Win32_Service instance from an SWbemServices object created with a moniker*

```
1:<?xml version="1.0"?>
.:
8:<package>
9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
28:    Set objWMIOutParameters = GetObject("WinMgmts:{impersonationLevel=impersonate, " & _
29:                                        "AuthenticationLevel=default, " & _
30:                                        "(RemoteShutdown, Security)}!\\" & _
31:                                        cComputerName & "\" & cWMINameSpace).ExecMethod _
32:                                        (cWMIClass & "='" & cWMIInstance & "'", "StopService")
33:
34:    If objWMIOutParameters.ReturnValue = 0 Then
35:       Wscript.Echo "Synchronous method execution successful."
36:    Else
37:       Wscript.Echo "Synchronous method execution failed."
```

```
38:    End If
..:
42:    ]]>
43:    </script>
44:  </job>
45:</package>
```

As we can see, line 28 immediately returns an **SWbemObject** object containing the return code of the invoked method. A connection a to particular WMI namespace on a remote computer is made only in one single executed line with the moniker (lines 28 to 32). The required impersonation and authentication levels with some privileges are set to create an **SWbemServices** object (lines 28 to 31). Next, a *Win32_Service* class instance is referenced and one method exposed by the *Win32_Service* class is executed from the selected instance (line 32).

4.4.4.5 The CIM class method qualifiers

Retrieving the method qualifiers is no different than the tactic used to retrieve the property qualifiers. Again, the principle is exactly the same. First, the script must retrieve the collection of methods exposed by a CIM class (stored in **SWbemMethodSet** object) and perform a loop to enumerate every method in the collection. Each method is stored in an **SWbemMethod** object. The object containing the collection of qualifiers defining the methods is, as usual, an **SWbemQualifierSet** object. This object contains every defined qualifier in an **SWbemQualifier** object. As we look to the method qualifiers, the **SWbemQualifierSet** object must be retrieved with the *Qualifier_* property exposed by the **SWbemMethod** object (see Table 4.18). The **SWbemQualifierSet** and **SWbemQualifier** objects are represented by numbers 29 and 30 in the WMI object model shown in Figure 4.4. Sample 4.22 shows this logic.

Sample 4.22 *Retrieving the method qualifiers of a WMI class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <script language="VBScript" src="..\Functions\DisplayQualifiersFunction.vbs" />
14:
15:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:
17:    <script language="VBscript">
18:    <![CDATA[
..:
22:    Const cComputerName = "LocalHost"
23:    Const cWMINameSpace = "root/cimv2"
24:    Const cWMIClass = "Win32_Service"
..:
```

```
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:
34:    Set objWMIInstance = objWMIServices.Get (cWMIClass, wbemFlagUseAmendedQualifiers)
35:
36:    Wscript.Echo String (60, 45)
37:    Wscript.Echo "Class Qualifiers:"
38:
39:    WScript.Echo "   " & objWMIInstance.Path_.RelPath
40:    DisplayQualifiers objWMIInstance.Qualifiers_, 0
41:
42:    Wscript.Echo String (60, 45)
43:    Wscript.Echo "Method Qualifiers:"
44:
45:    Set objWMIMethodSet = objWMIInstance.Methods_
46:    For Each objWMIMethod In objWMIMethodSet
47:        WScript.Echo "   " & objWMIMethod.Name
48:        DisplayQualifiers objWMIMethod.Qualifiers_, 0
49:    Next
..:
54:    ]]>
55:    </script>
56:  </job>
57:</package>
```

After retrieving the class instance (line 34), the script shows the class qualifiers with the help of a function enumerating and showing the properties of every qualifier (see Sample 4.16 ["Retrieving the property qualifiers of a WMI class"]). Once completed, the script retrieves the list of methods exposed by the class (lines 45 to 49). For every method, the script invokes the function to display the qualifier information (line 48). Here is a sample output for the *StartService* and *StopService* methods.

```
Method Qualifiers:
  StartService
    Name: Description
      Value:                  The StartService method attempts to place the service into its startup
                              state. It returns one of the following integer values:
                              0 - The request was accepted.
                              1 - The request is not supported.
                              2 - The user did not have the necessary access.
                              3 - The service cannot be stopped because other services that are
                                  running are dependent on it.
                              4 - The requested control code is not valid, or it is unacceptable to
                                  the service.
                              5 - The requested control code cannot be sent to the service because
                                  the state of the service (Win32_BaseService:State) is equal to 0,
                                  1, or 2.
                              6 - The service has not been started.
                              7 - The service did not respond to the start request in a timely
                                  fashion.
                              8 - Unknown failure when starting the service.
                              9 - The directory path to the service executable was not found.
                              10 - The service is already running.
                              11 - The database to add a new service is locked.
                              12 - A dependency for which this service relies on has been removed
```

```
                                from the system.
                           13 - The service failed to find the service needed from a dependent
                                service.
                           14 - The service has been disabled from the system.
                           15 - The service does not have the correct authentication to run on
                                the system.
                           16 - This service is being removed from the system.
                           17 - There is no execution thread for the service.
                           18 - There are circular dependencies when starting the service.
                           19 - There is a service running under the same name.
                           20 - There are invalid characters in the name of the service.
                           21 - Invalid parameters have been passed to the service.
                           22 - The account which this service is to run under is either invalid
                                or lacks the permissions to run the service.
                           23 - The service exists in the database of services available from the
                                system.
                           24 - The service is currently paused in the system.
  Amended:                 True
  Local:                   True
  Overridable:             True
  Propagates to instance: False
  Propagates to subclass: True
Name: MappingStrings
  Value:                   Win32API|Service Functions|StartService
  Amended:                 False
  Local:                   False
  Overridable:             True
  Propagates to instance: False
  Propagates to subclass: True
Name: Override
  Value:                   StartService
  Amended:                 False
  Local:                   False
  Overridable:             True
  Propagates to instance: False
  Propagates to subclass: True
Name: Values
  Value:                   Success
  Value:                   Not Supported
  Value:                   Access Denied
  Value:                   Dependent Services Running
  Value:                   Invalid Service Control
  Value:                   Service Cannot Accept Control
  Value:                   Service Not Active
  Value:                   Service Request Timeout
  Value:                   Unknown Failure
  Value:                   Path Not Found
  Value:                   Service Already Running
  Value:                   Service Database Locked
  Value:                   Service Dependency Deleted
  Value:                   Service Dependency Failure
  Value:                   Service Disabled
  Value:                   Service Logon Failed
  Value:                   Service Marked For Deletion
  Value:                   Service No Thread
  Value:                   Status Circular Dependency
  Value:                   Status Duplicate Name
  Value:                   Status Invalid Name
  Value:                   Status Invalid Parameter
  Value:                   Status Invalid Service Account
```

```
    Value:                  Status Service Exists
    Value:                  Service Already Paused
    Amended:                True
    Local:                  True
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
StopService
  Name: Description
    Value:                  The StopService method places the service in the stopped state. It
                            returns an integer value of 0 if the service was successfully stopped,
                            1 if the request is not supported, and any other number to indicate an
                            error.
    Amended:                True
    Local:                  True
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: MappingStrings
    Value:                  Win32API|Service Functions|ControlService|dwControl|SERVICE_CONTROL_STOP
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: Override
    Value:                  StopService
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
```

4.4.4.6 *The CIM class method parameters*

A CIM class may expose some properties and methods, but a method may also have some input and output parameters. By using the WMI scripting API, it is possible to retrieve the parameters associated to a method. Sample 4.23 executes this task.

Sample 4.23 *Retrieving the method properties with their qualifiers of a WMI class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <script language="VBScript" src="..\Functions\DisplayQualifiersFunction.vbs" />
14:
15:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
16:
17:    <script language="VBscript">
18:    <![CDATA[
..:
22:    Const cComputerName = "LocalHost"
23:    Const cWMINameSpace = "root/cimv2"
24:    Const cWMIClass = "Win32_Service"
```

```
..:
32:    On Error Resume Next
33:
34:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
35:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
36:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
37:
38:    Set objWMIInstance = objWMIServices.Get (cWMIClass, wbemFlagUseAmendedQualifiers)
39:
40:    Wscript.Echo String (60, 45)
41:    Wscript.Echo "Class Qualifiers:"
42:
43:    WScript.Echo "   " & objWMIInstance.Path_.RelPath
44:    DisplayQualifiers objWMIInstance.Qualifiers_, 0
45:
46:    Wscript.Echo String (60, 45)
47:    Wscript.Echo "Method Qualifiers:"
48:
49:    Set objWMIMethodSet = objWMIInstance.Methods_
50:    For Each objWMIMethod In objWMIMethodSet
51:        WScript.Echo "   " & objWMIMethod.Name
52:        DisplayQualifiers objWMIMethod.Qualifiers_, 0
53:
54:        Set objWMIObject = objWMIMethod.InParameters
55:        WScript.Echo "   " & objWMIObject.Path_.RelPath
56:        DisplayQualifiers objWMIObject.Qualifiers_, 2
57:
58:        Set objWMIPropertySet = objWMIObject.Properties_
59:        If Err.Number = 0 Then
60:           For Each objWMIProperty In objWMIPropertySet
61:               WScript.Echo "     " & objWMIProperty.Name
62:               DisplayQualifiers objWMIProperty.Qualifiers_, 4
63:           Next
64:        Else
65:           Err.Clear
66:        End If
..:
71:        Set objWMIObject = objWMIMethod.OutParameters
72:        WScript.Echo "   " & objWMIObject.Path_.RelPath
73:        DisplayQualifiers objWMIObject.Qualifiers_, 2
74:
75:        Set objWMIPropertySet = objWMIObject.Properties_
76:        If Err.Number = 0 Then
77:           For Each objWMIProperty In objWMIPropertySet
78:               WScript.Echo "     " & objWMIProperty.Name
79:               DisplayQualifiers objWMIProperty.Qualifiers_, 4
80:           Next
81:        Else
82:           Err.Clear
83:        End If
..:
88:    Next
..:
93:    ]]>
94:    </script>
95:  </job>
96:</package>
```

Basically, the script is the same as Sample 4.22 ("Retrieving the method qualifiers of a WMI class"). It first retrieves the CIM class with its qualifiers (line 44). Next, it retrieves the method collection available (line 49). While enumerating the method collection in a "For Each" loop (lines 50 to 88), the script displays the qualifiers associated with each method (line 52) and retrieves the associated parameters:

- With the *InParameters* property of the **SWbemMethod** object (see Table 4.18) for the input parameters (line 54)
- With the *OutParameters* property of the **SWbemMethod** object (see Table 4.18) for the output parameters (line 71)

Each of these properties returns an **SWbemObject** object whose properties define the input parameters or the output parameters of the executed method. If we look at the WMI object model presented in Figure 4.4, this implies that these two used properties of the **SWbemMethod** object (located as number 28 on the Figure 4.4) produce a new **SWbemObject** object that will be located as number 20 in the same figure. As this **SWbemObject** object represents the method parameters, it is possible to retrieve the qualifiers of such object. For this, the script uses the object number 21 of the WMI object model to retrieve the qualifier collection. This is performed on line 56 for the input parameters and line 73 for the output parameters.

As with any **SWbemObject** object, it is possible to retrieve a collection of properties and methods with their associated qualifiers (see Table 4.10). Of course, it is not sure that every method defined for a CIM class will have input parameters. Input or output parameters can be retrieved with the *Properties_* property exposed by the **SWbemObject** object. In such a case an **SWbemPropertySet** object containing a collection of **SWbemProperty** objects is returned. This is achieved on line 58 for the input parameters and line 75 for the output parameters. If some input or output parameters exist, then the returned **SWbemPropertySet** object can be enumerated. In this case, the script uses objects 23 and 24 in the WMI object model (see Figure 4.4).

4.4.4.7 The CIM class method parameter qualifiers

Because input or output parameters also have some characteristics, such as syntax, some qualifiers are also defined. When enumerating the input parameters (lines 60 to 63) and the output parameters (lines 77 to 80), the script invokes the function displaying the associated qualifiers (lines 62 and 79). At this stage of the script, we are using objects 25 and 26 in the WMI object model (see Figure 4.4) as a method parameter is represented by an

SWbemProperty object. Finally, once run, the script will output a lot of information. Below, a partial set of the retrieved information is visible for the *ChangeStartMode* method of the *Win32_Service* class with all input and output parameters and all associated qualifiers:

```
ChangeStartMode
  Name: Description
    Value:                  The ChangeStartMode method modifies the StartMode of a service. It
                            returns an integer value of 0 if the service was successfully modified,
                            1 if the request is not supported, and any other number to indicate an
                            error.
    Amended:                True
    Local:                  True
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
  Name: MappingStrings
    Value:                  Service Functions|ChangeServiceConfig|dwStartType
    Amended:                False
    Local:                  False
    Overridable:            True
    Propagates to instance: False
    Propagates to subclass: True
__PARAMETERS
    Name: abstract
      Value:                  True
      Amended:                False
      Local:                  True
      Overridable:            True
      Propagates to instance: False
      Propagates to subclass: False
  StartMode
      Name: CIMTYPE
        Value:                  string
        Amended:                False
        Local:                  True
        Overridable:            True
        Propagates to instance: True
        Propagates to subclass: True
      Name: ID
        Value:                  0
        Amended:                False
        Local:                  True
        Overridable:            False
        Propagates to instance: True
        Propagates to subclass: False
      Name: In
        Value:                  True
        Amended:                False
        Local:                  True
        Overridable:            True
        Propagates to instance: False
        Propagates to subclass: False
      Name: MappingStrings
        Value:                  Win32API|Service Structures|QUERY_SERVICE_CONFIG|dwStartType
        Amended:                False
        Local:                  True
        Overridable:            True
```

```
        Propagates to instance: False
        Propagates to subclass: True
      Name: ValueMap
        Value:                  Boot
        Value:                  System
        Value:                  Automatic
        Value:                  Manual
        Value:                  Disabled
        Amended:                False
        Local:                  True
        Overridable:            True
        Propagates to instance: False
        Propagates to subclass: True
__PARAMETERS
    Name: abstract
      Value:                  True
      Amended:                False
      Local:                  True
      Overridable:            True
      Propagates to instance: False
      Propagates to subclass: False
  ReturnValue
      Name: CIMTYPE
        Value:                  uint32
        Amended:                False
        Local:                  True
        Overridable:            True
        Propagates to instance: True
        Propagates to subclass: True
      Name: out
        Value:                  True
        Amended:                False
        Local:                  True
        Overridable:            True
        Propagates to instance: False
        Propagates to subclass: False
```

If you run the script, you will notice that a lot of information scrolls on the screen. At the end of the chapter, based on the structure developed here and with the previous scripts, we will create a script that loads the CIM class information into an Excel sheet. This makes the retrieved information easy to review. But for now, there is another important topic to look at: the WMI object path retrieved from the WMI scripting API.

4.4.5 Retrieving WMI object information with its WMI path

At the beginning of this chapter, we saw that the WMI object path is an important parameter for locating objects in the CIM repository. Added to this, the WMI moniker completes the set of information required to establish the connection to a real-world manageable entity. The WMI scripting API offers various ways to navigate through the CIM classes. From one class it is possible to retrieve many classes based on their relationships with each

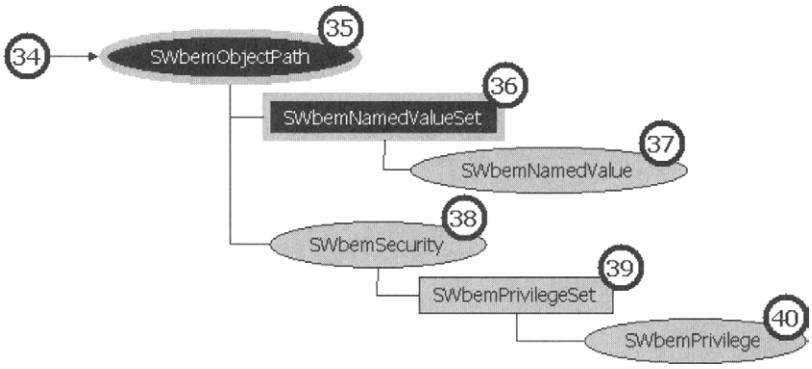


Figure 4.5
The WMI object model (3/3).

other. As classes have a relation, the instances of the classes also have the same kind of relation. At a certain time, after moving from one class to another, or from one instance to another, it would be interesting to retrieve the WMI object path of the examined class or instance. For this **SWbemObject** exposes a property called *Path_*. This property retrieves a new object called **SWbemObjectPath**. This object is represented in Figure 4.4 with the number 34.

Since this object contains some other objects, the WMI object model organization for the **SWbemObjectPath** object is represented in Figure 4.5.

Let's take a look at Sample 4.24. The script retrieves the WMI object path information.

Sample 4.24 *Retrieving the WMI object path properties of a WMI class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIPrivilegeSet = objWMILocator.Security_.Privileges
33:    objWMIPrivilegeSet.AddAsString "SeRemoteShutdownPrivilege", True
34:    objWMIPrivilegeSet.AddAsString "SeSecurityPrivilege", True
35:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
36:
```

```
37:     Set objWMIInstance = objWMIServices.Get (cWMIClass)
38:
39:     Set objWMIPath = objWMIInstance.Path_
40:
41:     WScript.Echo "Authority: " & objWMIPath.Authority
42:     WScript.Echo "Class: " & objWMIPath.Class
43:     WScript.Echo "DisplayName: " & objWMIPath.DisplayName
44:     WScript.Echo "IsClass: " & objWMIPath.IsClass
45:     WScript.Echo "IsSingleton: " & objWMIPath.IsSingleton
46:
47:     Set objWMINamedValueSet = objWMIPath.Keys
48:     WScript.Echo "Keys#: " & objWMINamedValueSet.Count
49:     For Each objWMINamedValue In objWMINamedValueSet
50:         WScript.Echo "  " & objWMINamedValue.Name & ": " & objWMINamedValue.Value
51:     Next
52:
53:     WScript.Echo "Locale: " & objWMIPath.Locale
54:     WScript.Echo "Namespace: " & objWMIPath.Namespace
55:     WScript.Echo "ParentNamespace: " & objWMIPath.ParentNamespace
56:     WScript.Echo "Path: " & objWMIPath.Path
57:     WScript.Echo "Relpath: " & objWMIPath.Relpath
58:
59:     WScript.Echo "Security_ (AuthenticationLevel): " & _
60:                     objWMIPath.Security_.AuthenticationLevel
61:     WScript.Echo "Security_ (ImpersonationLevel): " & _
62:                     objWMIPath.Security_.ImpersonationLevel
63:
64:     Set objWMIPrivilegeSet = objWMIPath.Security_.Privileges
65:     For Each objWMIPrivilege In objWMIPrivilegeSet
66:         WScript.Echo "  " & objWMIPrivilege.Name & ": " & _
67:                             objWMIPrivilege.Identifier & " (" & _
68:                             objWMIPrivilege.IsEnabled & ")."
69:         WScript.Echo "  ->" & objWMIPrivilege.DisplayName
70:     Next
71:
72:     WScript.Echo "Server: " & objWMIPath.Server
...:
81:     ]]>
82:     </script>
83: </job>
84:</package>
```

Initially, the script establishes a connection to a WMI object (lines 30 to 35). This connection is established with some privileged settings (lines 33 and 34). These privileges are set for academic purposes, as they are not required for the operation to be performed. Once done, the script retrieves the **SWbemObjectPath** object (line 39). Next, the script displays the values of every property (lines 41 to 57) exposed by the **SWbemObjectPath** object in Table 4.19. Note that some properties return objects. This is the case for the *Keys* property (line 47). This property returns an **SWbemNamedValueSet** object, which is a collection of **SWbemNamedValue** objects. These two objects are located as numbers 36 and 37, respectively, in the object model shown in Figure 4.5. The methods and the properties exposed are available in Tables 4.20 and 4.21.

Table 4.19 *The SWbemObjectPath Object*

Properties	Authority	String that defines the Authority component of the object path.
	Class	Name of the class that is part of the object path.
	DisplayName	String that contains the path in a form that can be used as a moniker display name. See Object Creation and Monikers.
	IsClass	Boolean value that indicates if this path represents a class. This is analogous to the __Genus property in the COM API.
	IsSingleton	Boolean value that indicates if this path represents a singleton instance.
	Keys	An SWbemNamedValueSet object that contains the key value bindings.
	Locale	String containing the locale for this object path.
	Namespace	Name of the namespace that is part of the object path. This is the same as the __Namespace property in the COM API.
	ParentNamespace	Name of the parent of the namespace that is part of the object path.
	Path	Contains the absolute path. This is the same as the __Path system property in the COM API. This is the default property of this object.
	Relpath	Contains the relative path. This is the same as the __Relpath system property in the COM API.
	Security_	Used to read or change the security settings.
	Server	Name of the server. This is the same as the __Server system property in the COM API.
Methods	SetAsClass ()	Forces the path to address a WMI class.
	SetAsSingleton ()	Forces the path to address a singleton WMI instance.

Table 4.20 *The SWbemNamedValueSet Object*

Properties	Count	The number of items in the collection.
Methods	Add strName, varVal, [iFlags = 0]	Adds an SWbemNamedValue object to the collection.
	Clone ()	Makes a copy of this SWbemNamedValueSet collection.
	DeleteAll ()	Removes all items from the collection, making the SWbemNamedValueSet object empty.
	Item strName, [iFlags = 0]	Retrieves an SWbemNamedValue object from the collection. This is the default method of the object.
	Remove strName, [iFlags = 0]	Removes an SWbemNamedValue object from the collection.

Table 4.21 *The SWbemNamedValue Object*

Properties	Name	Name of the SWbemNamedValue item.
	Value	Value of the SWbemNamedValue item. This is the default property of this object.

The second object returned is an **SWbemSecurity** object (lines 59 to 64). This object is exactly the same as the one we saw associated to the **SWbemLocator** object in the beginning of this chapter. In Figure 4.5, the **SWbemSecurity** object and its child objects are represented with numbers 38 to 40. When the script is executed with a connection to a class defined in the CIM repository, the output is as follows:

```
 1:  Authority:
 2:  Class: Win32_Service
 3:  DisplayName: WINMGMTS:{authenticationLevel=pktPrivacy,impersonationLevel=impersonate,
 4:                         (Security,RemoteShutdown)}!\\W2K-DPEN6400\ROOT\CIMV2:Win32_Service
 5:  IsClass: True
 6:  IsSingleton: False
 7:  Keys#: 0
 8:  Locale:
 9:  Namespace: ROOT\CIMV2
10:  ParentNamespace: ROOT
11:  Path: \\W2K-DPEN6400\ROOT\CIMV2:Win32_Service
12:  Relpath: Win32_Service
13:  Security_ (AuthenticationLevel): 6
14:  Security_ (ImpersonationLevel): 3
15:    SeSecurityPrivilege: 7 (True).
16:    ->Manage auditing and security log
17:    SeRemoteShutdownPrivilege: 23 (True).
18:    ->Force shutdown from a remote system
19:  Server: W2K-DPEN6400
```

In this output, notice that the *DisplayName* (lines 3 and 4) is nothing more than the WMI moniker to access the class definition. As we access a class, we also see that the *IsClass* property is set to True (line 5). During the connection, the script sets some security settings and privileges with the **SWbemLocator** object in order to have something to display when examining the **SWbemSecurity** object linked with the **SWbemObjectPath** (lines 13 to 18). Notice also that the script is able to retrieve more informative text about the privilege itself (line 15 and 18). In the first line of the display output, you can see that the *Authority* property doesn't return anything. This is normal, as no authority has been set to establish the WMI connection. Another interesting thing is the *Relpath* property. This property simply returns the relative path of the class, which is nothing more than the name of the class (line 12). As we examine a class definition, there is no real-world manageable object instantiated. This is why there is no key listed (line 7). If we change a few lines (lines 22 and 36) in the object connection to connect

to a real-world manageable object, the script connection part becomes as follows:

```
..:
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
31:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
32:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
33:    Set objWMIPrivilegeSet = objWMILocator.Security_.Privileges
34:    objWMIPrivilegeSet.AddAsString "SeRemoteShutdownPrivilege", True
35:    objWMIPrivilegeSet.AddAsString "SeSecurityPrivilege", True
36:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
37:
38:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
..:
..:
```

This script is basically the same as Sample 4.24 except that a WMI instance is retrieved. In this case, the output is:

```
 1:Authority:
 2:Class: Win32_Service
 3:DisplayName: WINMGMTS:{authenticationLevel=pktPrivacy,impersonationLevel=impersonate,
                (Security,RemoteShutdown)}!\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="SNMP"
 4:IsClass: False
 5:IsSingleton: False
 6:Keys#: 1
 7:  Name: SNMP
 8:Locale:
 9:Namespace: root\cimv2
10:ParentNamespace: root
11:Path: \\W2K-DPEN6400\root\cimv2:Win32_Service.Name="SNMP"
12:Relpath: Win32_Service.Name="SNMP"
13:Security_ (AuthenticationLevel): 6
14:Security_ (ImpersonationLevel): 3
15:  SeSecurityPrivilege: 7 (True).
16:  ->Manage auditing and security log
17:  SeRemoteShutdownPrivilege: 23 (True).
18:  ->Force shutdown from a remote system
19:Server: W2K-DPEN6400
```

The output is the same as before, except that we can notice three major differences:

1. Now that the script addresses a real-world instance (the SNMP *Win32_Service*), the *IsClass* property is set to False (line 4).
2. There is a key value (line 7).
3. The *DisplayName*, *Path*, and *Relpath* properties include the name of the referred instance.

4.4.6 Working with partial instances

Throughout the samples, each time a script retrieves an object instances all properties available from this object instance are returned. With WMI, it is possible to retrieve only the properties that are needed. In many cases, it is more expensive to retrieve all the properties of an instance than to retrieve just a few properties. WMI providers that support partial-instance operations can be much faster and use fewer computer resources because they only need to set or retrieve specific properties.

4.4.6.1 Reading partial instances

Some of the classes in the **Root\CIMv2** namespace that use the *CIMWin32* provider support partial-instance operations. It is important to note that WMI does not support partial-instance operations on instances that reside in the CIM repository and are not associated with a provider. Partial-instance operations are supported only when a provider supplies the instance. It is not a requirement for a provider to support partial-instance operations. In such a case, a provider may do the following:

- Provide more properties than the client requested
- Support partial-instance operations for some classes but not others
- Support partial-instance retrieval but not partial-instance update, or vice versa

Requesting a partial-instance operation is done by setting specific values in an **SWBemNamedValueSet** object. This **SWBemNamedValueSet** object is passed when retrieving the instance. The specific values set in **SWBemNamedValueSet** are summarized in Table 4.22.

Sample 4.25 shows how to proceed for partial instance retrieval.

Sample 4.25 *Retrieving one partial instance of the Win32_Service class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:    <object progid="WbemScripting.SWbemNamedValueSet" id="objWMINamedValueSet" />
15:
16:    <script language="VBscript">
17:    <![CDATA[
..:
21:    Const cComputerName = "LocalHost"
22:    Const cWMINameSpace = "root/cimv2"
23:    Const cWMIClass = "Win32_Service"
```

```
24:    Const cWMIInstance = "SNMP"
..:
31:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
32:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
33:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
34:
35:    objWMINamedValueSet.Add "__GET_EXTENSIONS", True
36:    objWMINamedValueSet.Add "__GET_EXT_CLIENT_REQUEST", True
37:    objWMINamedValueSet.Add "__GET_EXT_PROPERTIES", Array ("Name", "State")
38:
39:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'", _
40:                                             , _
41:                                             objWMINamedValueSet)
42:
43:    Set objWMIPropertySet = objWMIInstance.Properties_
44:    For Each objWMIProperty In objWMIPropertySet
45:        If Not IsNull (objWMIProperty.Value) Then
46:           If objWMIProperty.IsArray Then
47:              For Each varElement In objWMIProperty.Value
48:                  WScript.Echo "  " & objWMIProperty.Name & " (" & varElement & ")"
49:              Next
50:           Else
51:              WScript.Echo "  " & objWMIProperty.Name & " (" & objWMIProperty.Value & ")"
52:           End If
53:        End If
54:    Next
..:
60:    ]]>
61:    </script>
62:  </job>
63:</package>
```

If you compare Sample 4.25 with Sample 4.13 ("Retrieving the property collection of a WMI instance"), you will notice a few changes. The first modification is the instantiation of the **SWBemNamedValueSet** object (line 14). The second important modification is the content initialization of the **SWBemNamedValue** object contained in the **SWBemNamedValueSet** (lines 35 to 37). These values are used as an input for the WMI provider to inform how a partial-instance operation must be processed. Line 35 specifies that a partial-instance operation must be performed, line 36 informs WMI that information stored in the **SWBemNamedValueSet** is on request from the WMI client, and line 37 lists the properties to be retrieved. If we execute the script, only 14 properties will be retrieved. Curiously, we listed only two properties to retrieve. Providers that support partial-instance retrieval may return more properties than the client requests while not returning the complete instance. This relies on the WMI provider implementation. Sample 4.25 illustrates this behavior. If Sample 4.25 is executed without a partial-instance reading, it retrieves 24 properties. However, with the partial-instance feature, only 13 properties are retrieved, even if only two properties are specified in the **SWbemNamedValueSet** object. This

Table 4.22 *Partial Instance Keyword Commands*

Instance Retrieval Named Values	
__GET_EXTENSIONS	Boolean value set to TRUE. Used to signal that partial-instance retrieval operations are being used. If any of the other values are used, this one must be set.
__GET_EXT_PROPERTIES	Array of strings listing the properties to be retrieved. Cannot be simultaneously specified with __GET_EXT_KEYS_ONLY.
__GET_EXT_KEYS_ONLY	Boolean value set to TRUE. Indicates that only key(s) should be provided in the returned object. Cannot be simultaneously specified with __GET_EXT_PROPERTIES.
__GET_EXT_CLIENT_REQUEST	Boolean value set to TRUE. Indicates that the caller was the one who wrote the value into the context object and that it was not propagated from another dependent operation.
Instance Update Named Values	
__PUT_EXTENSIONS	Boolean value set to TRUE. A value indicating that one or more of the other context values has been specified.
__PUT_EXT_STRICT_NULLS	Boolean value set to TRUE. Indicates that the client has intentionally set properties to Null and expects the write operation to succeed. If the provider cannot set the values to NULL, an error should be reported.
__PUT_EXT_PROPERTIES	Array of strings that contains a list of property names to be updated. May be used alone or in combination with __PUT_EXT_PROPERTIES. The values are, of course, in the instance being written.
__PUT_EXT_ATOMIC	Boolean value set to TRUE. Indicates that all updates must succeed simultaneously (atomic semantics) or the provider must revert back. There can be no partial success. May be used alone or in combination with other flags.
__PUT_EXT_CLIENT_REQUEST	Boolean value set to TRUE. Set by the client during the initial request. Used to prevent reentrancy errors.

demonstrates the partial-instance retrieval feature and the limitation imposed by the WMI provider.

Now, if line 37

```
37:    objWMINamedValueSet.Add "__GET_EXT_PROPERTIES", Array ("Name", "State")
```

is modified as follows

```
37:    objWMINamedValueSet.Add "__GET_EXT_KEYS_ONLY", True
```

the script retrieves only the *Key* properties of the instance. You must use one *NamedValue* or the other. These two *NamedValues* cannot be combined in

the same **SWbemNamedValueSet**. Only four methods from the **SWBemServices** object that retrieve a single instance support the partial-instance retrieval. These methods are:

1. *Get*
2. *GetAsync*
3. *InstancesOf*
4. *InstancesOfAsync*

4.4.6.2 Updating partial instances

Occasionally, you may want to update only part of an instance. For example, some instances have a large number of properties, or the system you are writing to does not update an instance entirely. Updating a large number of instances may reduce system performance. Therefore, it is possible to update only a part of an instance, and thus reduce the amount of information sent to WMI. However, WMI does not directly support partial-instance operations, nor do most providers. Therefore, if you write an application that uses partial-instance operations, be prepared for your calls to fail with either the **wbemErrProviderNotCapable** or **wbemErrNotSupported** error codes. Currently, the classes and the WMI provider we have worked with until now do not support the partial-instance update. However, this functionality will be ubiquitous when you work with the *SNMP* and *Active Directory* WMI providers.

In scripting, this operation is necessary only to aid performance when updating one or two writeable properties in a very large number of objects over an enterprise. Otherwise the normal invocations of the *Put_* or *PutAsync_* methods, while seeming to write the entire object, are actually only updating the properties that the provider has write-enabled. To perform a partial-instance update, the coding is exactly the same as the partial-instance retrieval. The difference resides in the keyword (see Table 4.22) used in the **SWbemNamedValueSet** object. For instance, the code snippet below illustrates the coding technique:

```
1:  objWMINamedValueSet.Add "__PUT_EXTENSIONS", True
2:  objWMINamedValueSet.Add "__PUT_EXT_CLIENT_REQUEST", True
3:  objWMINamedValueSet.Add "__PUT_EXT_PROPERTIES", Array ("Property1, Property2")
4:
5:  objWMIInstance.Put_ wbemChangeFlagUpdateOnly Or wbemFlagReturnWhenComplete, _
6:                      objWMINamedValueSet
```

Note the presence of the wbemChangeFlagUpdateOnly flag when invoking the *Put_* method of the **SWbemObject** object. If it is not specified, the method invocation will fail.

4.4.7 Retrieving WMI object system properties

When discovering Sample 4.13 ("Retrieving the property collection of a WMI instance"), we used an **SWbemPropertySet** collection object initialized with the *Properties_* property of the **SWbemObject**. Although this method retrieves the properties of an **SWbemObject** instance, it does not retrieve the system properties of the object. Under Windows XP and Windows.NET Server, the **SWbemObject** object exposes a property called *SystemProperties_* (see Table 4.10), returning an **SWbemPropertySet** object, which contains a collection of **SWbemProperty** objects representing system properties. This *SystemProperties_* property usage is illustrated in Sample 4.26.

Sample 4.26 *Retrieving the system properties of an SWbemObject*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
34:
35:    Set objWMIPropertySet = objWMIInstance.SystemProperties_
36:    For Each objWMIProperty In objWMIPropertySet
37:        If objWMIProperty.IsArray Then
38:           For Each varElement In objWMIProperty.Value
39:               WScript.Echo "  " & objWMIProperty.Name & " (" & varElement & ")"
40:           Next
41:        Else
42:           WScript.Echo "  " & objWMIProperty.Name & " (" & objWMIProperty.Value & ")"
43:        End If
44:    Next
..:
50:    ]]>
51:    </script>
52:  </job>
53:</package>
```

The *SystemProperties_* property reference at line 35 retrieves the list of system properties. Except for this small change, the script is exactly the

same as that in Sample 4.13 ("Retrieving the property collection of a WMI instance"). The output of Sample 4.26 is as follows:

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

  __PATH (\\PPC284346\root\cimv2:Win32_Service.Name="SNMP")
  __NAMESPACE (root\cimv2)
  __SERVER (PPC284346)
  __DERIVATION (Win32_BaseService)
  __DERIVATION (CIM_Service)
  __DERIVATION (CIM_LogicalElement)
  __DERIVATION (CIM_ManagedSystemElement)
  __PROPERTY_COUNT (25)
  __RELPATH (Win32_Service.Name="SNMP")
  __DYNASTY (CIM_ManagedSystemElement)
  __SUPERCLASS (Win32_BaseService)
  __CLASS (Win32_Service)
  __GENUS (2)
```

In this sample output, we see that the derivation hierarchy of the class is available. Although it is possible to use this method to retrieve the parent classes of the examined class (Sample 4.26 examines the *Win32_Service* class), the **SWbemObject** object exposes a more specialized property called *Derivation_*. Sample 4.27 shows how to use this property.

Sample 4.27 *Retrieving the parent classes (superclasses) of an SWbemObject*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
15:    <script language="VBscript">
16:    <![CDATA[
..:
20:    Const cComputerName = "LocalHost"
21:    Const cWMINameSpace = "root/cimv2"
22:    Const cWMIClass = "Win32_Service"
23:    Const cWMIInstance = "SNMP"
..:
30:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
31:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
32:    Set objWMIServices = objWMILocator.ConnectServer(cComputerName, cWMINameSpace, "", "")
33:    Set objWMIInstance = objWMIServices.Get (cWMIClass & "='" & cWMIInstance & "'")
34:
35:    arrayParentClasses = objWMIInstance.Derivation_
36:    For Each varElement In arrayParentClasses
37:        WScript.Echo "  " & varElement
38:    Next
..:
```

```
43:    ]]>
44:    </script>
45:  </job>
46:</package>
```

The main difference of the *Derivation_* property is that it returns an array that contains the list of parent classes instead of a collection stored in an **SWbemPropertySet** object. The first element in the array defines the superclass and the last element defines the dynasty class (the top class of the hierarchy). A sample output for the *Win32_Service* class follows:

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

  Win32_BaseService
  CIM_Service
  CIM_LogicalElement
  CIM_ManagedSystemElement
```

4.5 Browsing the namespaces in the CIM repository

Throughout this chapter, we have worked in the **Root\CIMv2** namespace. Although it is an important namespace because it contains most of the WMI Win32 class representing Windows manageable entities, there are other namespaces that contain quite useful information. As previously mentioned, instead of listing all namespaces and classes in tables, it is much more interesting to have a script to retrieve such information. This protects against any new non-documented WMI extensions, because a script can retrieve this information from the CIM repository for you. Note, as already mentioned, using a WQL query to retrieve all namespaces available is not possible because WQL does not support cross-namespace queries.

Since the classes are contained in namespaces, let's retrieve the namespaces first. In Chapter 2, we listed in Table 2.1 the namespaces available in the CIM repository. Sample 4.28 shows how to retrieve the namespaces available in the CIM repository.

Sample 4.28 *Browsing all namespaces defined in the CIM repository*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
14:
```

```
 15:    <script language="VBscript">
 16:    <![CDATA[
 ..:
 20:    Const cComputerName = "LocalHost"
 21:    Const cWMINameSpace = "Root"
 22:    Const cWMINamespaceClass = "__NAMESPACE"
 23:
 24:    DisplayNameSpaces cWMINameSpace
 25:
 26:    ' ------------------------------------------------------------------------
 27:    Function DisplayNameSpaces (ByVal strWMINameSpace)
 ..:
 32:        objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
 33:        objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
 34:        Set objWMIServices = objWMILocator.ConnectServer(cComputerName, strWMINameSpace, "", "")
 35:
 36:        Set objWMINSInstances = objWMIServices.InstancesOf (cWMINamespaceClass,
wbemQueryFlagShallow)
 37:
 38:        For Each objWMINSInstance in objWMINSInstances
 39:            WScript.Echo strWMINameSpace & "/" & objWMINSInstance.Name
 40:            DisplayNameSpaces strWMINameSpace & "/" & objWMINSInstance.Name
 41:        Next
 ..:
 47:    End Function
 48:
 49:    ]]>
 50:    </script>
 51:  </job>
 52:</package>
```

This script is a quite easy script as it uses just a few of the WMI scripting API objects mentioned in this chapter. Basically, the script starts to browse from the root namespace (line 21) by invoking a recursive function called DisplayNameSpaces() (line 24). This function performs a connection to the CIM repository with the **SWbemLocator** object (line 34). Once connected, the script requests the instances of the *__NAMESPACE* system class with the *InstancesOf* method of the **SWbemServices** object (line 36). As this method retrieves a collection, the script enumerates in a For Each loop the namespace collection. For instance, the script output when executed on a Windows.NET Server system is:

```
C:\>BrowseNameSpaceWithAPI.wsf
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

Root/CIMV2
Root/CIMV2/ms_409
Root/Cli
Root/DEFAULT
Root/DEFAULT/ms_409
Root/directory
Root/directory/LDAP
Root/directory/LDAP/ms_409
```

```
Root/Microsoft
Root/Microsoft/HomeNet
Root/MicrosoftActiveDirectory
Root/MicrosoftADStatus
Root/MicrosoftCluster
Root/MicrosoftDNS
Root/MicrosoftIISv2
Root/MicrosoftIISv2/ms_409
Root/MicrosoftNLB
Root/MSAPPS
Root/Policy
Root/Policy/ms_409
Root/RSOP
Root/RSOP/Computer
Root/RSOP/Computer/ms_409
Root/RSOP/User
Root/RSOP/User/ms_409
Root/RSOP/User/S_1_5_21_2025429265_507921405_1202660629_1112
Root/RSOP/User/S_1_5_21_2025429265_507921405_1202660629_1113
Root/RSOP/User/S_1_5_21_2025429265_507921405_1202660629_500
Root/Security
Root/Security/SCE
Root/snmp
Root/snmp/localhost
Root/subscription
Root/subscription/ms_409
Root/WMI
Root/WMI/ms_409
```

If we reuse Sample 4.28 with some minor modifications based on our WMI object model knowledge, we can create a script that browses the WMI name spaces and locates instances of a given class. This is the purpose of Sample 4.29.

Sample 4.29 *Browsing the namespaces to find instances of a given class*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
14:      <unnamed name="Class" helpstring="the WMI Class instances to retrieve across all namespaces."
                               required="true" type="string" />
15:      <named name="Machine" helpstring="determine the WMI system to connect to. (default=LocalHost)"
                               required="false" type="string"/>
16:      <named name="User" helpstring="determine the UserID to perform the remote connection.
                               (default=none)" required="false" type="string"/>
17:      <named name="Password" helpstring="determine the password to perform the remote connection.
                               (default=none)" required="false" type="string"/>
18:    </runtime>
19:
20:    <script language="VBScript" src="..\Functions\DisplayInstanceProperties.vbs" />
21:    <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
```

```
22:
23:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
24:
25:    <script language="VBscript">
26:    <![CDATA[
..:
30:    Const cComputerName = "LocalHost"
31:    Const cWMINameSpace = "Root"
32:    Const cWMINamespaceClass = "__NAMESPACE"
33:    Const cWMIClass = "__EventProviderRegistration"
..:
39:    ' ------------------------------------------------------------------------------
40:    ' Parse the command line parameters
41:    If WScript.Arguments.Unnamed.Count = 0 Then
42:       strWMIClass = InputBox ("Enter the WMI Class to examine: ", _
43:                               "WMI Class:", _
44:                               cWMIClass)
45:
46:       If Len (strWMIClass) = 0 Then
47:          WScript.Arguments.ShowUsage()
48:          WScript.Quit
49:       End If
50:    Else
51:       strWMIClass = WScript.Arguments.Unnamed.Item(0)
52:    End If
53:       strWMIClass = Ucase (strWMIClass)
54:
55:    strUserID = WScript.Arguments.Named("User")
56:    If Len(strUserID) = 0 Then strUserID = ""
57:
58:    strPassword = WScript.Arguments.Named("strPassword")
59:    If Len(strPassword) = 0 Then strPassword = ""
60:
61:    strComputerName = WScript.Arguments.Named("Machine")
62:    If Len(strComputerName) = 0 Then strComputerName = cComputerName
63:
64:    DisplayNameSpaces cWMINameSpace, strWMIClass, strUserID, strPassword, strComputerName
65:
66:    ' ------------------------------------------------------------------------------
67:    Function DisplayNameSpaces (ByVal strWMINameSpace, _
68:                                ByVal strWMIClass, _
69:                                ByVal strUserID, _54:
70:                                ByVal strPassword, _
71:                                ByVal strComputerName)
..:
81:       objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
82:       objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
83:       Set objWMIServices = objWMILocator.ConnectServer(strComputerName, strWMINameSpace,
                                                           strUserID, strPassword)
84:       If Err.Number Then ErrorHandler (Err)
85:
86:       Set objWMIInstances = objWMIServices.InstancesOf (strWMIClass, wbemQueryFlagShallow)
87:       If Err.Number Then
88:          Err.Clear
89:       Else
90:          WScript.Echo strWMINameSpace
91:          For Each objWMIInstance in objWMIInstances
92:              DisplayProperties objWMIInstance, 2
93:          Next
94:          Set objWMIInstances = Nothing
```

```
 95:         End If
 96:
 97:        Set objWMINSInstances = objWMIServices.InstancesOf (cWMINamespaceClass, wbemQueryFlagShallow)
 98:         For Each objWMINSInstance in objWMINSInstances
 99:             DisplayNameSpaces strWMINameSpace & "/" & objWMINSInstance.Name, _
100:                               strWMIClass, strUserID, strPassword, strComputerName
101:         Next
...:
106:     End Function
107:
108:     ]]>
109:     </script>
110:   </job>
111:</package>
```

The script uses the same structure as Sample 4.28. The addition concerns the retrieval of the instances of a given class. By default, the script retrieves the instances of the *__EventProviderRegistration* system class across all namespaces. Any other class can be provided, as the script accepts another class name from its command line parameters. For this, the script uses the WSH XML command line parsing features (lines 41 to 62) explained in Chapter 1. Next, the script calls the same recursive function (line 64) as Sample 4.28. In the recursive function, the script connects to the examined namespace (line 83) and instantiates the given class at line 86. If no error occurs, the script enumerates the instances (lines 91 to 93) and lists their related properties (lines 92). The DisplayProperties() function, displaying the instance properties, encapsulates the same logic used throughout this chapter to explore the **SWBemPropertySet** object. This is nothing new from a WMI scripting API point of view; it is just a different organization used to retrieve information (see Sample 4.30 below). Note that the routine adds information about the syntax used by the property (line 33, 38, 45, 53, 61, 66, and 74) and converts any Date/Time value to a readable format by using a new WMI object called **SWBemDateTime** (line 50). We will revisit this object in the next chapter. The routine is shown in Sample 4.30.

Sample 4.30 *A generic routine to display the SWbemPropertySet object*

```
.:
.:
.:
 6:' ------------------------------------------------------------------------------------------
 7:Function DisplayProperties (objWMIInstance, intIndent)
..:
14:     Set objWMIPropertySet = objWMIInstance.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:                For Each varElement In objWMIProperty.Value
32:                    WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
33:                                 " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ")"
34:                    DisplayProperties varElement, intIndent + 2
35:                Next
36:             Else
37:                WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
38:                             " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ")"
39:                DisplayProperties objWMIProperty.Value, intIndent + 2
40:             End If
41:          Else
42:             If objWMIProperty.IsArray Then
43:                For Each varElement In objWMIProperty.Value
44:                    WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
45:                                 " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ") = " & _
46:                                 varElement
47:                Next
48:             Else
49:                If objWMIProperty.Name = "TIME_CREATED" Then
50:                   objWMIDateTime.SetFileTime (objWMIProperty.Value)
51:
52:                   WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
53:                                " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ") = " & _
54:                                objWMIDateTime.GetVarDate (True) & _
55:                                " (" & objWMIDateTime.Value & ")
56:                Else
57:                   If objWMIProperty.CIMType = wbemCimtypeDatetime Then
58:                      objWMIDateTime.Value = objWMIProperty.Value
59:
60:                      WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
61:                                   " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ") = " &
62:                                   objWMIDateTime.GetVarDate (True) & _
63:                                   " (" & objWMIProperty.Value & ")"
64:                   Else
65:                      WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
66:                                   " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ") = " &
67:                                   objWMIProperty.Value
68:                   End If
69:                End If
70:             End If
71:          End If
72:       Else
73:          WScript.Echo Space (intIndent) & strCIMKey & objWMIProperty.Name & _
74:                       " (" & GetCIMSyntaxText (objWMIProperty.CIMType) & ") = (null)"
75:       End If
76:    Next
..:
79:End Function
80:
```

```
 81:' ------------------------------------------------------------------------------------------
 82:Function GetCIMSyntaxText (intCIMType)
 83:
 84:    Select Case intCIMType
 85:           ' Signed 16-bit integer
 86:           Case 2
 87:                GetCIMSyntaxText = "wbemCimtypeSint16"
 88:           ' Signed 32-bit integer
 89:           Case 3
 90:                GetCIMSyntaxText = "wbemCimtypeSint32"
 91:           ' 32-bit real number
 92:           Case 4
 93:                GetCIMSyntaxText = "wbemCimtypeReal32"
 94:           ' 64-bit real number
 95:           Case 5
 96:                GetCIMSyntaxText = "wbemCimtypeReal64"
 97:           ' String
 98:           Case 8
 99:                GetCIMSyntaxText = "wbemCimtypeString"
100:           ' Boolean value
101:           Case 11
102:                GetCIMSyntaxText = "wbemCimtypeBoolean"
...:
124:           ' Date/time value
125:           Case 101
126:                GetCIMSyntaxText = "wbemCimtypeDatetime"
127:           ' Reference to a CIM object.
128:           Case 102
129:                GetCIMSyntaxText = "wbemCimtypeReference"
130:           ' 16-bit character
131:           Case 103
132:                GetCIMSyntaxText = "wbemCimtypeChar16"
133:    End Select
134:
135:End Function
```

There are two particularities in this routine. First, if the property syntax is an object (**wbemCimtypeObject**), then the routine is executed recursively to examine the properties of the object (lines 34 and 39). This ensures that an object is encapsulated in a property (as will be the case when we capture WMI events from scripts in Chapter 6) to display the object's properties content. Second, if the property name is *TIME_CREATED* or if its syntax is a time (**wbemCimtypeDatetime**), the property value is stored in **SWbemDateTime** object. An **SWbemDateTime** object is a helper object designed to convert the DMTF time format into some more readable forms.

The following output shows the results obtained when executing Sample 4.30, if we are searching for all instances of the *Win32_WMISetting* class. Notice that the property information mentions the CIM syntax.

```
C:\>BrowseNameSpaceForInstancesWithAPI.wsf Win32_WMISetting
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.
```

```
Root/CIMV2
  ASPScriptDefaultNamespace (wbemCimtypeString) = root\cimv2
  ASPScriptEnabled (wbemCimtypeBoolean) = (null)
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\cimwin32.mof
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\cimwin32.mfl
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\system.mof
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\wmipcima.mof
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\wmipcima.mfl
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\regevent.mof
  ..................
  ..................
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\wbem\ADStatus\TrustMon.mof
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\System32\replprov.mof
  AutorecoverMofs (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\MOF\good\msioff9.mof
  AutorecoverMofs (wbemCimtypeString) = J:\Program Files\Microsoft Platform SDK\Bin\WMI\Ev
  AutoStartWin9X (wbemCimtypeUint32) = (null)
  BackupInterval (wbemCimtypeUint32) = 30
  BackupLastTime (wbemCimtypeDatetime) = 13-05-2001 15:24:28
  BuildVersion (wbemCimtypeString) = 2462.0000
  Caption (wbemCimtypeString) = (null)
  DatabaseDirectory (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\Repository
  DatabaseMaxSize (wbemCimtypeUint32) = (null)
  Description (wbemCimtypeString) = (null)
  EnableAnonWin9xConnections (wbemCimtypeBoolean) = (null)
  EnableEvents (wbemCimtypeBoolean) = True
  EnableStartupHeapPreallocation (wbemCimtypeBoolean) = False
  HighThresholdOnClientObjects (wbemCimtypeUint32) = 20000000
  HighThresholdOnEvents (wbemCimtypeUint32) = 20000000
  InstallationDirectory (wbemCimtypeString) = J:\WINDOWS\system32\WBEM
  LastStartupHeapPreallocation (wbemCimtypeUint32) = (null)
  LoggingDirectory (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\Logs\
  LoggingLevel (wbemCimtypeUint32) = 1
  LowThresholdOnClientObjects (wbemCimtypeUint32) = 10000000
  LowThresholdOnEvents (wbemCimtypeUint32) = 10000000
  MaxLogFileSize (wbemCimtypeUint32) = 65536
  MaxWaitOnClientObjects (wbemCimtypeUint32) = 60000
  MaxWaitOnEvents (wbemCimtypeUint32) = 2000
  MofSelfInstallDirectory (wbemCimtypeString) = J:\WINDOWS\system32\WBEM\MOF
  SettingID (wbemCimtypeString) = (null)
```

Another variation of the script is shown in Sample 4.31. It uses basically the same logic as Sample 4.29 ("Browsing the namespaces to find instances of a given class"), but instead of retrieving the class instances, it retrieves the class definition across namespaces. The code is similar to that shown in Sample 4.29, but instead of using the *InstancesOf* method of the **SWbemServices** object, it uses the *Get* method of the **SWbemServices** object (line 85) with logic modified accordingly.

Sample 4.31 *Browsing the namespaces to find class definitions*

```
..:
..:
..:
82:        objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
83:        objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
```

```
  84:        Set objWMIServices = objWMILocator.ConnectServer(strComputerName, strWMINameSpace,
                                                              strUserID, strPassword)
  85:        If Err.Number Then ErrorHandler (Err)
  86:
  87:        Set objWMIInstance = objWMIServices.Get (strWMIClass)
  88:        If Err.Number Then
  89:           Err.Clear
  90:        Else
  91:           WScript.Echo strWMINameSpace
  92:           DisplayProperties objWMIInstance, 2
  93:           Set objWMIInstance = Nothing
  94:        End If
  95:
  96:       Set objWMINSInstances = objWMIServices.InstancesOf (cWMINamespaceClass, wbemQueryFlagShallow)
  97:        For Each objWMINSInstance in objWMINSInstances
  98:            DisplayNameSpaces strWMINameSpace & "/" & objWMINSInstance.Name, _
  99:                              strWMIClass, strUserID, strPassword, strComputerName
 100:        Next
 101:        Set objWMINSInstances = Nothing
 102:
 103:        Set objWMIServices = Nothing
 ...:
 ...:
 ...
```

The following output shows the results obtained when executing Sample 4.31 if we are searching for the *Win32_WMISetting* class. Notice that the property value is empty, as we retrieve the class definition and not its instance.

```
C:\>BrowseNameSpaceForClassWithAPI.wsf Win32_WMISetting
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Root/CIMV2
  ASPScriptDefaultNamespace (wbemCimtypeString) = \\root\cimv2
  ASPScriptEnabled (wbemCimtypeBoolean) = (null)
  AutorecoverMofs (wbemCimtypeString) = (null)
  AutoStartWin9X (wbemCimtypeUint32) = (null)
  BackupInterval (wbemCimtypeUint32) = (null)
  BackupLastTime (wbemCimtypeDatetime) = (null)
  BuildVersion (wbemCimtypeString) = (null)
  Caption (wbemCimtypeString) = (null)
  DatabaseDirectory (wbemCimtypeString) = (null)
  DatabaseMaxSize (wbemCimtypeUint32) = (null)
  Description (wbemCimtypeString) = (null)
  EnableAnonWin9xConnections (wbemCimtypeBoolean) = (null)
  EnableEvents (wbemCimtypeBoolean) = (null)
  EnableStartupHeapPreallocation (wbemCimtypeBoolean) = (null)
  HighThresholdOnClientObjects (wbemCimtypeUint32) = (null)
  HighThresholdOnEvents (wbemCimtypeUint32) = (null)
  InstallationDirectory (wbemCimtypeString) = (null)
  LastStartupHeapPreallocation (wbemCimtypeUint32) = (null)
  LoggingDirectory (wbemCimtypeString) = (null)
  LoggingLevel (wbemCimtypeUint32) = (null)
  LowThresholdOnClientObjects (wbemCimtypeUint32) = (null)
  LowThresholdOnEvents (wbemCimtypeUint32) = (null)
  MaxLogFileSize (wbemCimtypeUint32) = (null)
```

```
  MaxWaitOnClientObjects (wbemCimtypeUint32) = (null)
  MaxWaitOnEvents (wbemCimtypeUint32) = (null)
  MofSelfInstallDirectory (wbemCimtypeString) = (null)
  SettingID (wbemCimtypeString) = (null)
Root/CIMV2/ms_409
  ASPScriptDefaultNamespace (wbemCimtypeString) = (null)
  ASPScriptEnabled (wbemCimtypeBoolean) = (null)
  AutorecoverMofs (wbemCimtypeString) = (null)
  AutoStartWin9X (wbemCimtypeUint32) = (null)
  BackupInterval (wbemCimtypeUint32) = (null)
  BackupLastTime (wbemCimtypeDatetime) = (null)
  BuildVersion (wbemCimtypeString) = (null)
  Caption (wbemCimtypeString) = (null)
  DatabaseDirectory (wbemCimtypeString) = (null)
  DatabaseMaxSize (wbemCimtypeUint32) = (null)
  Description (wbemCimtypeString) = (null)
  EnableAnonWin9xConnections (wbemCimtypeBoolean) = (null)
  EnableEvents (wbemCimtypeBoolean) = (null)
  EnableStartupHeapPreallocation (wbemCimtypeBoolean) = (null)
  HighThresholdOnClientObjects (wbemCimtypeUint32) = (null)
  HighThresholdOnEvents (wbemCimtypeUint32) = (null)
  InstallationDirectory (wbemCimtypeString) = (null)
  LastStartupHeapPreallocation (wbemCimtypeUint32) = (null)
  LoggingDirectory (wbemCimtypeString) = (null)
  LoggingLevel (wbemCimtypeUint32) = (null)
  LowThresholdOnClientObjects (wbemCimtypeUint32) = (null)
  LowThresholdOnEvents (wbemCimtypeUint32) = (null)
  MaxLogFileSize (wbemCimtypeUint32) = (null)
  MaxWaitOnClientObjects (wbemCimtypeUint32) = (null)
  MaxWaitOnEvents (wbemCimtypeUint32) = (null)
  MofSelfInstallDirectory (wbemCimtypeString) = (null)
  SettingID (wbemCimtypeString) = (null)
```

4.6 A script to explore the CIM repository

As we explore WMI we discover many tables describing the methods and the properties exposed by the different objects that constitute the WMI scripting API. Besides the WMI scripting API features list, there are also the CIM class definitions. The only CIM class definition really used (up to now) is the *Win32_Service* class shown in Table 4.1. WMI offers more than 600 CIM classes. As the number of CIM classes is quite huge, it is impossible to list all the properties and methods exposed by these classes.

Based on the materials covered in this chapter, it is much more valuable to think about a script retrieving the interesting information about any CIM classes. Because everything resides in the CIM repository, it is possible to produce readable output of that content. This is the purpose of the next sample, which summarizes the miscellaneous WMI scripting APIs we have discovered.

What do we mean by "readable output"? Since there is a lot of information to display, a good place to load this information is in an Excel sheet.

This facilitates the review of the information once the script has been executed. Which information is to loaded in the Excel sheet? The script retrieves all information related to the CIM class definitions. This implies the following:

- The CIM class with its related qualifiers
- The CIM class properties with their related qualifiers
- The CIM class methods with their related qualifiers
- The CIM class method parameters with their related qualifiers
- The CIM classes derived from a CIM class

Of course, retrieving all the information in a single run is probably too much in some cases. A nice thing would be to specify a script input parameter that defines the level of information to be retrieved. This filters the retrieved information. Doing so, the script user would be able to select the quantity of information to load into the Excel sheet.

The script performing this task is displayed in Sample 4.32. Before examining the script code, let's see how the script can be used and the type of result it will produce. This will help to understand the code. First, the script makes extensive use of the WSH features for a Windows Script File (see Chapter 1) to read the command-line parameters. Here is the Help output of the script input parameters:

```
C:\>LoadCIMinXL.Wsf /?
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.

Usage: LoadCIMinXL Class [/Machine:value][/NameSpace:value][/Level:value][/Sub[+|-]][/Origin[+|-]]

Options:

Class     : the WMI Class to explore.
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)
NameSpace : determine the WMI namespace to connect to. (default=Root\CIMv2
Level     : determine the depth of the CIM repository exploration. (default=5)
            1: Load the class name(s) only.
            2: Load the class name(s) with the class properties.
            3: Load the class name(s) with the class properties and methods.
            4: Load the class name(s) with the class properties, methods
               and method input parameters.
            5: Load the class name(s) with the class properties, methods
               and method input/output parameters.
            6: Load the class name(s) with the class properties, methods,
               method input/output parameters and qualifiers.
Sub       : Explore subclasses. (Default=False)
Origin    : Show properties and methods if examined class is the origin. (Default=False)
```

Because the purpose of the script is to examine a CIM class definition, the script requests an existing class name as a mandatory parameter. This is the purpose of the **/Class** parameter. It is now possible to specify another machine than the one on which the script is running. This optional parameter is a switch called **/Machine**. If you perform a connection to another system, the script proposes to specify a UserID and a password with the switches **/User** and **/Password**. By default, the script uses the current security context. The script examines the **Root\CIMv2** namespace, but it is possible to specify another WMI namespace by using the **/Namespace** switch. Next, it is possible to determine the amount of information to be retrieved with the **/Level** switch. Each level corresponds to a certain quantity of information to be retrieved from the CIM repository. Let's examine the different levels and what they retrieve:

- **Level 1:** This level retrieves only the class name from the CIM repository. This level is of interest only when the **/Sub** switch is enabled. In this case, if the class is derived, the script will load all the subclasses in an indented way. This shows the parent-child relationship between classes.
- **Level 2:** This level retrieves the information provided in level 1 plus the properties exposed by the CIM class. Note that the combination of the **/Sub** switch produces the output of the properties for the given class and the classes derived from that class.
- **Level 3:** This level retrieves the information provided by previous levels plus the methods exposed by the CIM class. If the **/Sub** switch is provided, the method of the derived class is also retrieved.
- **Level 4:** This level retrieves the information provided by previous levels plus the input parameters of the retrieved methods. The script behaves in the same way as before if the **/Sub** switch is given.
- **Level 5:** This level is the same as level 4, except that the script will also retrieve the output parameters returned by the methods associated with the CIM classes. The script behaves in the same way as before if the **/Sub** switch is given. This level is the default level used by the script if no level is specified.
- **Level 6:** This level retrieves the biggest set of information from the CIM repository. It retrieves all information given by previous levels plus all WMI qualifiers related to every property, method, and method parameter.

There is another switch we have not yet mentioned. This is the **/Origin** switch. Usually, the script retrieves the information from a CIM class even if

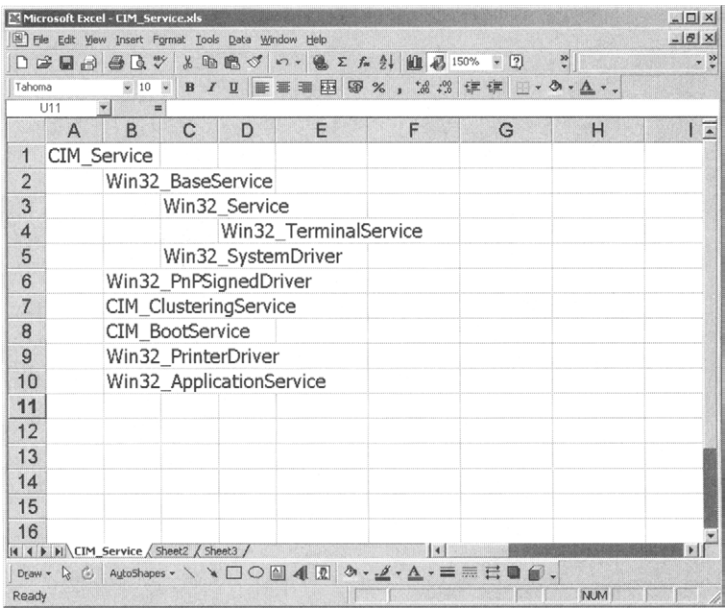


Figure 4.6
The CIM_Service class and its child classes loaded in an Excel sheet.

the considered CIM class has a property or a method inherited from a parent class. By specifying the **/Origin** switch, the script will retrieve the information associated with a class only if that information is directly defined in that class. In other words, if the property or the method is inherited from a parent class, the script will not display that information. This switch has a real interest if it is used in combination with the **/Sub** switch. During the examination of the parent class and any derived classes, the output shows where the properties and the methods are defined. Note that with level 6, the output contains information showing the origin of the related property or method (which class defines that property or method).

Let's examine an output sample with the following command line:

```
C:\>LoadCIMinXL.Wsf CIM_Service /Sub+ /Level:1
```

Figure 4.6 contains the output.

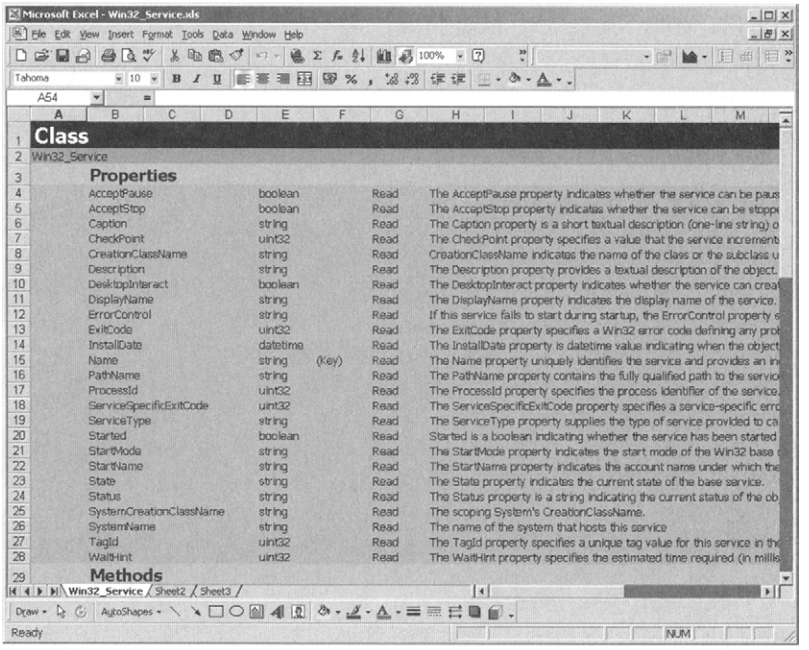
Of course, using Excel to load such information is not really interesting. But now, let's examine the output obtained by examining the *Win32_Service* class with the following command line:

```
C:\>LoadCIMinXL.Wsf Win32_Service /Machine:MyLocalMachine /Namespace:ROOT\CIMv2 /Level:5
```

This is the same as issuing the same command without parameters, since the previous command-line sample uses the defaults used by the script:

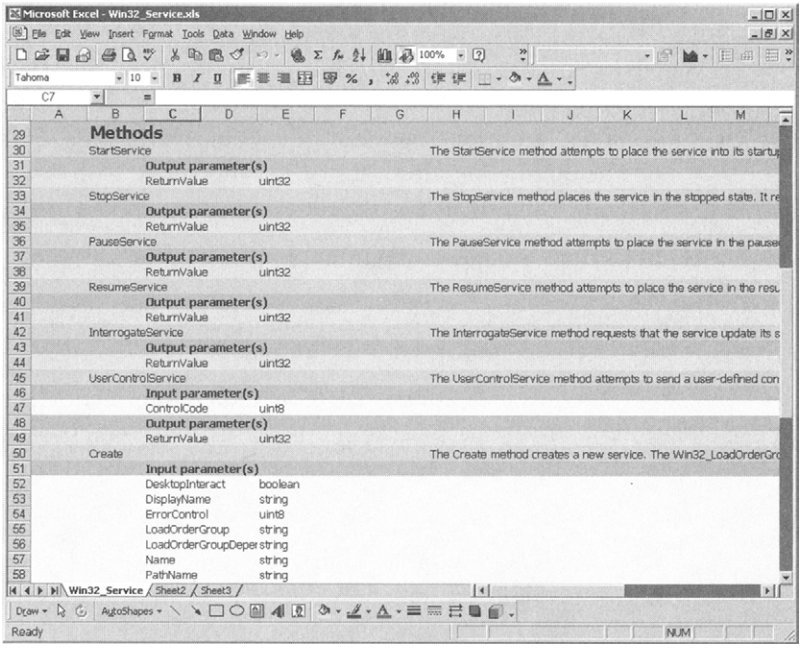
```
C:\>LoadCIMinXL.Wsf Win32_Service
```

Figure 4.7 *The Win32_Service class with its properties in an Excel sheet.*



Figures 4.7 and 4.8 contain the output.

Figure 4.8 *The Win32_Service class with its methods and methods parameters in an Excel sheet.*



The script makes extensive use of colors to clarify the distinction between the CIM class properties, the CIM class methods, and the CIM class method input and output parameters.

By using the following command line:

```
C:\>LoadCIMinXL.Wsf Win32_Service /Level:6
```

the script will retrieve the maximum information available. In this case, every qualifier available for every CIM class property, CIM class method, and CIM class method input and output parameter will be loaded in to the Excel sheet. Sample output is shown in Figure 4.9.

For instance, with the qualifiers displayed, it is easy to determine whether a CIM class is an association class or a singleton class. For the properties, the qualifiers inform you whether a CIM class property is read-only or read/write. The qualifier also determines whether the CIM class property is a key. Since the amount of information can be very large, Figure 4.9 can't

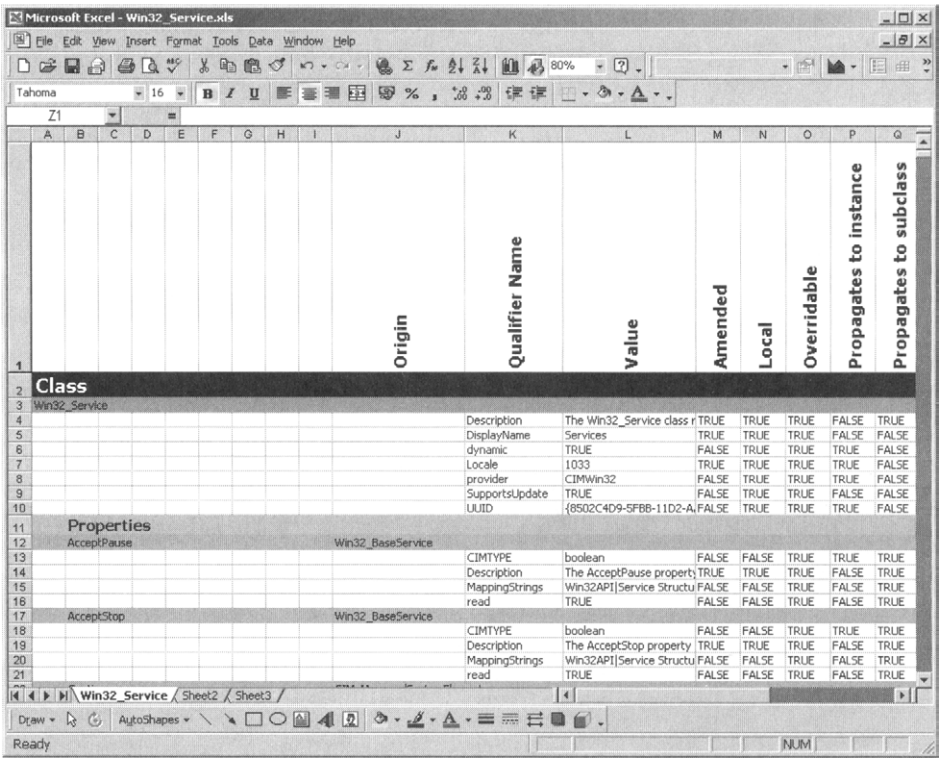


Figure 4.9 *The Win32_Service class with its properties, methods, and all existing qualifiers in an Excel sheet.*

display the complete output, but we recommend that you play with the script to discover its possibilities. Once you get used to it, you will realize the real possibilities offered by the script (as a real tool) to retrieve live information about a CIM class definition. This script will be helpful for all subsequent programming that can be performed on any class available from the CIM repository.

Now that we know what we can perform with the script, let's examine how the script is coded. It is only a brief examination as all of the materials used in this script have been discussed throughout this chapter. The script code is presented in Sample 4.32.

Since the data read from the CIM repository is loaded into an Excel sheet, the script performs a lot of Excel sheet formatting operations. Based on the nature of the information loaded (directly in relation with the **/Level** switch defined), the script formats the cell with a particular font size and background color. To get a complete understanding of Microsoft Excel object programming, you can refer to the Microsoft Office Programming SDK. If you don't have access to such a reference, you will see by reading the script that most of the statements are easy to understand.

Sample 4.32 *A Windows Script File self-documenting the CIM repository classes in an Excel sheet*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
..:
31:    </runtime>
32:
33:    <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
34:
35:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
36:    <object progid="EXCEL.application" id="objXL"/>
37:    <object progid="WScript.Shell" id="WshSHell"/>
38:
39:    <script language="VBscript">
40:    <![CDATA[
..:
44:    Const cLevelClassOnly = 1
45:    Const cLevelClassWithProps = 2
46:    Const cLevelClassWithPropsAndMethods = 3
47:    Const cLevelClassWithPropsAndMethodsWithInParams = 4
48:    Const cLevelClassWithPropsAndMethodsWithInOutParams = 5
49:    Const cLevelClassWithPropsQAndMethodsQWithParamsQ = 6
..:
63:    Const cComputerName = "LocalHost"
64:    Const cWMINameSpace = "Root/CIMv2"
```

```
..:
81:    ' ------------------------------------------------------------------------------
82:    ' Parse the command line parameters
83:    If WScript.Arguments.Unnamed.Count = 0 Then
84:       strWMIClass = InputBox ("Enter the WMI Class to examine: ", _
85:                               "WMI Class:", _
86:                               "Win32_Service")
87:
88:       If Len (strWMIClass) = 0 Then
89:          WScript.Arguments.ShowUsage()
90:          WScript.Quit
91:       End If
92:    Else
93:       strWMIClass = WScript.Arguments.Unnamed.Item(0)
94:    End If
95:    ' strWMIClass = Ucase (strWMIClass)
96:
97:    strUserID = WScript.Arguments.Named("User")
98:    If Len(strUserID) = 0 Then strUserID = ""
99:
100:   strPassword = WScript.Arjuments.Named(Password:)
101:   If Len(strPassword) = 0 Then strPassword = ""
102:
103:   strComputerName = WScript.Arguments.Named("Machine")
104:   If Len(strComputerName) = 0 Then strComputerName = cComputerName
105:
106:   strWMINameSpace = WScript.Arguments.Named("NameSpace")
107:   If Len(strWMINameSpace) = 0 Then strWMINameSpace = cWMINameSpace

108:   strWMINameSpace = UCase (strWMINameSpace)
109:
110:   intExplorationDepth = Cint (WScript.Arguments.Named("Level"))
111:   If intExplorationDepth = 0 Then
112:      intExplorationDepth = cLevelClassWithPropsAndMethodsWithInOutParams
113:   End If
114:
115:   boolSubClass = WScript.Arguments.Named("Sub")
116:   boolOrigin = Not WScript.Arguments.Named("Origin")
117:
118:   ' ------------------------------------------------------------------------------
119:   ' Prepare an Excel worksheet
120:   ' Make it visible, don't hide it because for the save, the user will be prompted!
121:   objXL.Visible = True
122:
123:   ' Open Excel and start an empty workbook
124:   objXL.Workbooks.Add
125:
126:   ' Put the cursor on the A1 cell
127:   objXL.ActiveSheet.range("A1").Activate
128:   objXL.ActiveSheet.Name = Mid (strWMIClass, 1, 31)
129:
130:   objXL.Columns("A:Z").Font.Name = "Tahoma"
131:   objXL.Columns("A:Z").IndentLevel = 0
132:   objXL.Columns("A:Z").VerticalAlignment = 2
133:
134:   If intExplorationDepth > cLevelClassWithPropsAndMethodsWithInOutParams Then
135:
136:      ' Format the XL Sheet
137:      objXL.Columns("A:I").ColumnWidth = 5
138:      objXL.Columns("J").ColumnWidth = 22
```

```
139:        objXL.Columns("K").ColumnWidth = 16
140:        objXL.Columns("L").ColumnWidth = 22
141:        objXL.Columns("M:Z").ColumnWidth = 7
142:        objXL.Rows("1").Orientation = 90
143:        objXL.Rows("1").VerticalAlignment = 3
144:        objXL.Rows("1").HorizontalAlignment = 3
145:
146:        objXL.Range("A" & objXL.ActiveCell.Row & ":Z" & objXL.ActiveCell.Row).Font.Bold = True
147:        objXL.Range("A" & objXL.ActiveCell.Row & ":Z" & objXL.ActiveCell.Row).Font.Size = 16
148:
149:        intX = cOriginPosition
150:        ' Origin
151:        objXL.Activecell.Offset(0, intX).Value = "Origin"
152:
153:        intX = cQualifierPosition
154:        ' Name
155:        objXL.Activecell.Offset(0, intX).Value = "Qualifier Name"
156:        ' Value
157:        objXL.Activecell.Offset(0, intX + 1).Value = "Value"
158:        ' Amended
159:        objXL.Activecell.Offset(0, intX + 2).Value = "Amended"
160:        ' Local
161:        objXL.Activecell.Offset(0, intX + 3).Value = "Local"
162:        ' Overridable
163:        objXL.Activecell.Offset(0, intX + 4).Value = "Overridable"
164:        ' Propagates to instance
165:        objXL.Activecell.Offset(0, intX + 5).Value = "Propagates to instance"
166:        ' Propagates to subclass
167:        objXL.Activecell.Offset(0, intX + 6).Value = "Propagates to subclass"
168:
169:        intX = 0
170:        objXL.Activecell.Offset(1, 0).Activate
171:    End If
172:
173:    ' ------------------------------------------------------------------------------
174:    ' Connect to WMI
175:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
176:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
177:    Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
178:                                                     strWMINameSpace, _
179:                                                     strUserID, _
180:                                                     strPassword)
181:    If Err.Number Then ErrorHandler (Err)
182:
183:    DisplayClasses intX, objWMIServices, strWMIClass
...:
187:    ' ------------------------------------------------------------------------------
188:    objXL.ActiveSheet.range("A1").Activate
189:
190:    ' Save & Close the Workbook. If file exists, user will be prompted.
192:          (WshShell.CurrentDirectory & "\" & objXL.ActiveSheet.Name)
193:    objXL.Workbooks.close
194:    objXL.Quit
195:
196:    ' ------------------------------------------------------------------------------
197:    Function DisplayClasses (ByVal intX, ByVal objWMIServices, ByVal strWMIClass)
...:
209:        Set objWMIInstance = objWMIServices.Get (strWMIClass, wbemFlagUseAmendedQualifiers)
210:        If Err.Number Then ErrorHandler (Err)
211:
```

```
191:    objXL.Workbooks.Application.ActiveWorkbook.SaveAs _
212:        If intExplorationDepth > cLevelClassOnly Then
213:           ' Format the XL sheet
214:           objXL.Range(...).Font.Bold = True
215:           objXL.Range(...).Font.Size = 18

216:           objXL.Range(...).Font.ColorIndex = White1
217:           objXL.Range(...).Interior.ColorIndex = Black
218:
219:           ' List the classes
220:           ' Class
221:           objXL.Activecell.Offset(0, 0).Value = "Class"
222:           objXL.Activecell.Offset(1, 0).Activate
223:        End If
224:
225:        WScript.Echo Space (intX) & strWMIClass
226:        objXL.Activecell.Offset(0, intX).Value = strWMIClass
227:
228:        If intExplorationDepth > cLevelClassOnly Then
229:           ' Format the XL sheet
230:           objXL.Range(...).Interior.ColorIndex = Aqua1
231:        End If
232:
233:        If intExplorationDepth > cLevelClassWithPropsAndMethodsWithInOutParams Then
234:           DisplayQualifiers (objWMIInstance.Qualifiers_)
235:        End If
236:
237:        objXL.Activecell.Offset(1, 0).Activate
238:
239:        If intExplorationDepth > cLevelClassOnly Then
240:           Set objWMIPropertySet = objWMIInstance.Properties_
241:           If objWMIPropertySet.Count Then
242:              intX = intX + 1
243:
244:              ' Format the XL Sheet
245:              objXL.Range(...).Font.Bold = True
246:              objXL.Range(...).Font.Size = 14
247:              objXL.Range(...).Interior.ColorIndex = Silver
248:
249:              ' List the properties of the Class
250:              ' Properties
251:              objXL.Activecell.Offset(0, intX).Value = "Properties"
252:              objXL.Activecell.Offset(1, 0).Activate
253:
254:              For Each objWMIProperty In objWMIPropertySet
255:
256:                  If (objWMIProperty.Origin = strWMIClass) Or boolOrigin Then
257:                     objXL.Activecell.Offset(0, intX).Value = objWMIProperty.Name
258:                     objXL.Range(...).Interior.ColorIndex = Aqua2
259:                     If intExplorationDepth < cLevelClassWithPropsQAndMethodsQWithParamsQ Then
260:                        objXL.Activecell.Offset(0, intX + 3).Value =
                                        objWMIProperty.Qualifiers_.Item("CIMTYPE").Value
261:
262:                        boolCIMKey = objWMIProperty.Qualifiers_.Item("key").Value
263:                        If Err.Number Then
264:                           Err.Clear
265:                           boolCIMKey = False
266:                        End If
267:                        If boolCIMKey Then
268:                           objXL.Activecell.Offset(0, intX + 4).Value = "(Key)"
```

```
269:                        End If
270:
271:                        boolCIMRead = objWMIProperty.Qualifiers_.Item("read").Value
272:                        If Err.Number Then
273:                           Err.Clear
274:                           boolCIMRead = False
275:                        End If
276:
277:                        boolCIMWrite = objWMIProperty.Qualifiers_.Item("write").Value
278:                        If Err.Number Then
279:                           Err.Clear
280:                           boolCIMWrite = False
281:                        End If
282:
283:                        If boolCIMRead ANd boolCIMWrite Then
284:                           objXL.Activecell.Offset(0, intX + 5).Value = "Read/Write"
285:                        Else
286:                           If boolCIMRead Then
287:                              objXL.Activecell.Offset(0, intX + 5).Value = "Read"
288:                           End If
289:                           If boolCIMWrite Then
290:                              objXL.Activecell.Offset(0, intX + 5).Value = "Write"
291:                           End If
292:                        End If
293:
294:                        strDescription = objWMIProperty.Qualifiers_.Item("Description").Value
295:                        If Err.Number Then
296:                           Err.clear
297:                        Else
298:                           objXL.Activecell.Offset(0, intX + 6).Value = strDescription
299:                           objXL.Activecell.Offset(0, intX + 6).WrapText = False
300:                        End If
301:
302:                     End If
303:                     If intExplorationDepth >
                            cLevelClassWithPropsAndMethodsWithInOutParams Then
304:                        objXL.Activecell.Offset(0, cOriginPosition).Value =
                                        objWMIProperty.Origin
305:                        DisplayQualifiers (objWMIProperty.Qualifiers_)
306:                     End If
307:
308:                     objXL.Activecell.Offset(1, 0).Activate
309:                  End If
310:               Next
311:            End If
312:            Set objWMIPropertySet = Nothing
313:
314:            If intExplorationDepth > cLevelClassWithProps Then
315:               Set objWMIMethodSet = objWMIInstance.Methods_
316:               If objWMIMethodSet.Count Then

317:                  ' Format the XL Sheet
318:                  objXL.Range(...).Font.Bold = True
319:                  objXL.Range(...).Font.Size = 14
320:                  objXL.Range(...).Interior.ColorIndex = Silver
321:
322:                  ' List the methods of the Class
323:                  ' Properties
324:                  objXL.Activecell.Offset(0, intX).Value = "Methods"
325:                  objXL.Activecell.Offset(1, 0).Activate
```

```
326:
327:                For Each objWMIMethod In objWMIMethodSet
328:                    If (objWMIMethod.Origin = strWMIClass) Or boolOrigin Then
329:                       objXL.Activecell.Offset(0, intX).Value = objWMIMethod.Name
330:                       objXL.Range(...).Interior.ColorIndex = White2
331:                          If intExplorationDepth >
                                  cLevelClassWithPropsAndMethodsWithInOutParams Then
332:                          objXL.Activecell.Offset(0, cOriginPosition).Value =
                                          objWMIMethod.Origin
333:                          DisplayQualifiers (objWMIMethod.Qualifiers_)
334:                       Else
335:                          strDescription = objWMIMethod.Qualifiers_.Item("Description").Value
336:                          If Err.Number Then
337:                             Err.clear
338:                          Else
339:                             objXL.Activecell.Offset(0, intX + 6).Value = strDescription
340:                             objXL.Activecell.Offset(0, intX + 6).WrapText = False
341:                          End If
342:                       End If
343:                       objXL.Activecell.Offset(1, 0).Activate
344:
345:                       If intExplorationDepth > cLevelClassWithPropsAndMethods Then
346:                          Set objWMIObject = objWMIMethod.InParameters
347:                          Set objWMIPropertySet = objWMIObject.Properties_
348:                          If Err.Number = 0 Then
349:                             intX = intX + 1
350:
351:                             ' Format the XL Sheet
352:                             objXL.Range(...).Font.Bold = True
353:                             objXL.Range(...).Font.Size = 10
354:                             objXL.Range(...).Interior.ColorIndex = Silver
355:
356:                             ' List the methods of the Class
357:                             ' Parameters
358:                             objXL.Activecell.Offset(0, intX).Value = "Input parameter(s)"
359:                             If intExplorationDepth >
                                    cLevelClassWithPropsAndMethodsWithInOutParams Then
360:                                DisplayQualifiers (objWMIObject.Qualifiers_)
361:                             End If
362:                             objXL.Activecell.Offset(1, 0).Activate
363:
364:                             For Each objWMIProperty In objWMIPropertySet
365:
366:                                 If (objWMIProperty.Origin = strWMIClass) Or boolOrigin Then
367:                                    objXL.Activecell.Offset(0, intX).Value =
                                                objWMIProperty.Name
368:                                    objXL.Range(...).Interior.ColorIndex = White3
369:                                    If intExplorationDepth =
                                             cLevelClassWithPropsAndMethodsWithInParams Or _
370:                                       intExplorationDepth =
                                             cLevelClassWithPropsAndMethodsWithInOutParams Then
371:                                       objXL.Activecell.Offset(0, intX + 2).Value =
                                            objWMIProperty.Qualifiers_.Item("CIMTYPE").Value
372:                                       strDescription =
                                            objWMIProperty.Qualifiers_.Item("Description").Value
373:                                       If Err.Number Then
374:                                          Err.clear
375:                                       Else
376:                                          objXL.Activecell.Offset(0, intX + 5).Value =
                                                   strDescription
```

```
377:                                    objXL.Activecell.Offset(0, intX + 5).WrapText =
                                             False
378:                                 End If
379:                              End If
380:                              If intExplorationDepth >
                                    cLevelClassWithPropsAndMethodsWithInOutParams Then
381:                                 objXL.Activecell.Offset(0, cOriginPosition).Value =
                                          objWMIProperty.Origin
382:                                 DisplayQualifiers (objWMIProperty.Qualifiers_)
383:                              End If
384:                              objXL.Activecell.Offset(1, 0).Activate
385:                           End If
386:
387:                        Next
388:                        intX = intX - 1
389:                     Else
390:                        Err.Clear
391:                     End If
392:                     Set objWMIPropertySet = Nothing
393:                     Set objWMIObject = Nothing
394:                  End If
395:
396:                  If intExplorationDepth >
                        cLevelClassWithPropsAndMethodsWithInParams Then
397:                     Set objWMIObject = objWMIMethod.OutParameters
398:                     Set objWMIPropertySet = objWMIObject.Properties_
399:                     If Err.Number = 0 Then
400:                        intX = intX + 1
401:
402:                        ' Format the XL Sheet
403:                        objXL.Range(...).Font.Bold = True
404:                        objXL.Range(...).Font.Size = 10
405:                        objXL.Range(...).Interior.ColorIndex = Silver
406:
407:                        ' List the methods of the Class
408:                        ' Parameters
409:                        objXL.Activecell.Offset(0, intX).Value = "Output parameter(s)"
410:                        If intExplorationDepth >
                              cLevelClassWithPropsAndMethodsWithInOutParams Then
411:                           DisplayQualifiers (objWMIObject.Qualifiers_)
412:                        End If
413:                        objXL.Activecell.Offset(1, 0).Activate
414:
415:                        For Each objWMIProperty In objWMIPropertySet
416:
417:                            If (objWMIProperty.Origin = strWMIClass) Or boolOrigin Then
418:                               objXL.Activecell.Offset(0, intX).Value =
                                             objWMIProperty.Name
419:                               objXL.Range(...).Interior.ColorIndex = Yellow
420:                               If intExplorationDepth =
                                    cLevelClassWithPropsAndMethodsWithInOutParams Then
421:                                  objXL.Activecell.Offset(0, intX + 2).Value =
                                          objWMIProperty.Qualifiers_.Item("CIMTYPE").Value
422:                                  strDescription =
                                       objWMIProperty.Qualifiers_.Item("Description").Value
423:                                  If Err.Number Then
424:                                     Err.clear
425:                                  Else
426:                                     objXL.Activecell.Offset(0, intX + 5).Value =
                                             strDescription
```

```
427:                                        objXL.Activecell.Offset(0, intX + 5).WrapText =
                                                 False
428:                                     End If
429:                                  End If
430:                                  If intExplorationDepth >
                                         cLevelClassWithPropsAndMethodsWithInOutParams Then
431:                                     objXL.Activecell.Offset(0, cOriginPosition).Value =
                                              objWMIProperty.Origin
432:                                     DisplayQualifiers (objWMIProperty.Qualifiers_)
433:                                  End If
434:                                  objXL.Activecell.Offset(1, 0).Activate
435:                               End If
436:
437:                            Next
438:                            intX = intX - 1
439:                         Else
440:                            Err.Clear
441:                         End If
442:                         Set objWMIPropertySet = Nothing
443:                         Set objWMIObject = Nothing
444:                      End If
445:                   End If
446:
447:               Next
448:            End If
449:            Set objWMIMethodSet = Nothing
450:         End If451:
452:         Set objWMIInstance = Nothing
453:
454:         intX = intX - 1
455:      End If
456:
457:      If boolSubClass Then
458:         Set objWMISubClasses = objWMIServices.SubClassesOf (strWMIClass, _
459:                                                             wbemQueryFlagShallow)
460:         For Each objWMISubClass in objWMISubClasses
461:             DisplayClasses intX + 1, objWMIServices, objWMISubClass.Path_.RelPath
462:         Next
463:         Set objWMISubClasses = Nothing
464:      End If
465:
466:   End Function
467:
468:   ' ------------------------------------------------------------------------------
469:   Function DisplayQualifiers (objWMIQualifiers)
...:
478:      intX = cQualifierPosition
479:
480:      For Each objWMIQualifier In objWMIQualifiers
481:          objXL.Activecell.Offset(1, 0).Activate
482:          objXL.Activecell.Offset(0, intX).Value = objWMIQualifier.Name
483:          If IsArray (objWMIQualifier.Value) Then
484:             For Each varElement In objWMIQualifier.Value
485:                 If Len (objXL.Activecell.Offset(0, intX + 1).Value) = 0 Then
486:                    objXL.Activecell.Offset(0, intX + 1).Value = varElement
487:                 Else
488:                    objXL.Activecell.Offset(0, intX + 1).Value =
                                objXL.Activecell.Offset(0, intX + 1).Value & _
489:                                                              Chr(10) & _
490:                                                              varElement
```

```
491:                    End If
492:                    If Ucase (objWMIQualifier.Name) = "DESCRIPTION" Then
493:                       objXL.Activecell.Offset(0, intX + 1).WrapText = False
494:                    End If
495:                 Next
496:              Else
497:                 objXL.Activecell.Offset(0, intX + 1).Value = objWMIQualifier.Value
498:                 If Ucase (objWMIQualifier.Name) = "DESCRIPTION" Then
499:                    objXL.Activecell.Offset(0, intX + 1).WrapText = False
500:                 End If
501:              End if
502:
503:              ' Amended
504:              objXL.Activecell.Offset(0, intX + 2).Value = objWMIQualifier.IsAmended
505:              ' Local
506:              objXL.Activecell.Offset(0, intX + 3).Value = objWMIQualifier.IsLocal
507:              ' Overridable
508:              objXL.Activecell.Offset(0, intX + 4).Value = objWMIQualifier.IsOverridable
509:              ' Propagates to instance
510:              objXL.Activecell.Offset(0, intX + 5).Value = objWMIQualifier.PropagatesToInstance
511:              ' Propagates to subclass
512:              objXL.Activecell.Offset(0, intX + 6).Value = objWMIQualifier.PropagatesToSubclass
513:        Next
514:
515:    End Function
516:
517:    ]]>
518:    </script>
519:  </job>
520:</package>
```

Basically, the script contains three distinct sections:

1. **The initialization part (lines 40 to 183).** This section contains the variables and constants declaration (line 44 to 81), the command-line parameters reading with the defaults (lines 83 to 116), the Excel sheet creation and formatting (lines 120 to 171), and the WMI connection with the WMI scripting API (lines 175 to 181).

2. **The section loading the information in the Excel sheet.** This part of the script is contained in a subfunction called at line 183. Basically, this subfunction (lines 197 to 466) implements the same structure as the function created in Sample 4.8 ("Retrieving the class inheritance tree with the SWbemServices object") to examine the subclasses recursively (lines 458 to 463). Before examining the subclasses, the script also looks at the following:

   - The class qualifiers (line 234)
   - The class properties (lines 240 to 312)
   - The class property qualifiers (line 305)
   - The class methods (lines 315 to 449)

- The class method qualifiers (line 333)
- The class method input parameters (lines 346 to 393)
- The class method input parameter qualifiers (line 360)
- The class method output parameters (lines 397 to 443)
- The class method output parameter qualifiers (line 411)

Even if this part of the script looks more complex simply because of the presence of the Excel statements formatting and loading data in the Excel sheet, by looking carefully at the pure WMI code logic you will notice that the logic is exactly the same as that discovered in the previous scripts. This script combines everything together.

3. **The section containing the function displaying the qualifiers (lines 469 to 515).** Basically, this function is the same as the one used before, but instead of outputting to a DOS command-prompt screen, the data is loaded in to the created Excel sheet.

4.7 Summary

This chapter is the first to discuss WMI scripting. We thoroughly recommend that you practice this technology to achieve a better active knowledge of the WMI scripting API.

As we have seen, it is possible to instantiate CIM classes or real-world manageable entities (mapped on defined CIM classes) by using a moniker or a series of WMI scripting APIs. Moreover, the security can be set at different object levels of the WMI scripting API. Besides these two aspects, it is also possible to combine the WMI scripting API with WQL queries.

However, we didn't examine all the WMI scripting API capabilities. Throughout the various samples we mainly discovered how the object model is organized. There are many more things to discover. For instance, we must still discover how asynchronous WMI operations can be performed, how WMI integrates with ADSI, and how a WMI instance can be represented in XML. These techniques are explained in the following chapter.

Mastering the scripting possibilities of the WMI COM API is key to developing management scripts for a production environment. This is why we focus on the WMI COM object capabilities before discussing the application of this technology to the real world.