5

The Optional Windows Components and Application WMI Providers

5.1 Objective

Throughout Chapters 2, 3, and 4 of this book dedicated to WMI, we discovered a lot of classes with their associated WMI providers. This discovery was a long process and showed some of the infinite management capabilities offered by WMI. However, the WMI providers and classes discovered in these chapters only focused on the core Operating System components. At no time were optional components, such as Terminal Services or Internet Information Server, considered. Similarly, Microsoft desktop and server applications, such as Office 2000, Internet Explorer, Exchange 2000, and SQL Server 2000, were not considered. Last but not least, management software, such as Insight Manager, Microsoft Operation Manager, and HP OpenView, were not considered either. The aim of this chapter is to give a brief overview of the WMI capabilities of these various applications. Reading the previous chapters brought you the necessary background to master the WMI scripting techniques. This implies that the remaining challenge is to understand the WMI capabilities of these applications. This is why this chapter will describe the WMI capabilities supported by the applications, without a specific focus on the scripting technique itself. Therefore, we will not systematically develop scripts for each feature, since the scripts will use the exact same scripting techniques as the ones provided throughout the previous chapters.

5.2 WMI and some additional Windows services

5.2.1 Network Load-Balancing service

Available under Windows Server 2003 and Windows 2000 Server, the *Network Load-Balancing* provider consists of two WMI providers registered in

Table 5.1 *The Network Load-Balancing Providers Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
Network Load Balancing Providers																
NlbsNicProv	Root/MicrosoftNLB		X	X				X	X	X		X				
Microsoft|NLB_Provider|V1.0	Root/MicrosoftNLB		X	X				X	X	X	X	X		X		

the **Root\MicrosoftNLB** namespace: the *NlbsNicProv* provider, implemented as an instance and method provider, and the *Microsoft|NLB_Provider|V1.0* provider, also implemented as an instance and method provider (see Table 5.1).

These two WMI providers support the classes listed in Table 5.2.

Table 5.2 *The Network Load-Balancing Providers Classes*

Name	Type	Description
MicrosoftNLB_Cluster	Dynamic	Represents an instance of a Network Load-Balancing cluster. Only nodes that have remote control enabled contribute to the ClusterState property as reported in this class and respond to the methods invoked from this class.
MicrosoftNLB_ClusterSetting	Dynamic	Represents data that identifies the Network Load-Balancing cluster to which a node belongs.
MicrosoftNLB_ExtendedStatus	Dynamic	Used by the Network Load-Balancing provider to report error codes specific to Network Load Balancing.
MicrosoftNLB_Node	Dynamic	Represents an instance of a node within a Network Load-Balancing cluster.
MicrosoftNLB_NodeSetting	Dynamic	Represents the configuration data specific to a node.
MicrosoftNLB_PortRuleDisabled	Dynamic	Represents a port rule on a single node whose filtering mode is set to "Disable." Do not use this class unless you need to manage Windows 2000 clusters. Always use MicrosoftNLB_PortRuleEx for managing port rules if possible.
MicrosoftNLB_PortRuleEx	Dynamic	The MicrosoftNLB_PortRuleEx WMI class represents a port rule on a node. The provider will only return the instances for this class that correspond to the node upon which it resides. Consequently, to configure a node, the client must explicitly connect to that node
MicrosoftNLB_PortRuleFailover	Dynamic	Represents a Network Load-Balancing port rule set to single-host filtering mode. Do not use this class unless you need to manage Windows 2000 clusters. Always use MicrosoftNLB_PortRuleEx for managing port rules if possible.
MicrosoftNLB_PortRuleLoadbalanced	Dynamic	Represents a Network Load-Balancing port rule set to multiple-host filtering mode. Do not use this class unless you need to manage Windows 2000 clusters. Always use MicrosoftNLB_PortRuleEx for managing port rules if possible.
MicrosoftNLB_ClusterClusterSetting	Association	Associates an instance of the MicrosoftNLB_Cluster class to an instance of the MicrosoftNLB_ClusterSetting class.
MicrosoftNLB_NodeNodeSetting	Association	Associates an instance of the MicrosoftNLB_Node class to an instance of the MicrosoftNLB_NodeSetting class.
MicrosoftNLB_NodeSettingPortRule	Association	Associates an instance of the MicrosoftNLB_NodeSetting class to instances of classes derived from MicrosoftNLB_PortRule.
MicrosoftNLB_ParticipatingNode	Association	Associates an instance of the MicrosoftNLB_Cluster class with participating MicrosoftNLB_Node class instances.
MicrosoftNLB_PortRule	Abstract Class	MicrosoftNLB_PortRule is an abstract base class from which classes that represent port rules are derived. Do not use this class unless you need to manage Windows 2000 clusters. Use MicrosoftNLB_PortRuleEx.
NlbsNic	Dynamic	Allows the management of the NLB network adapter settings through a set of WMI static methods.

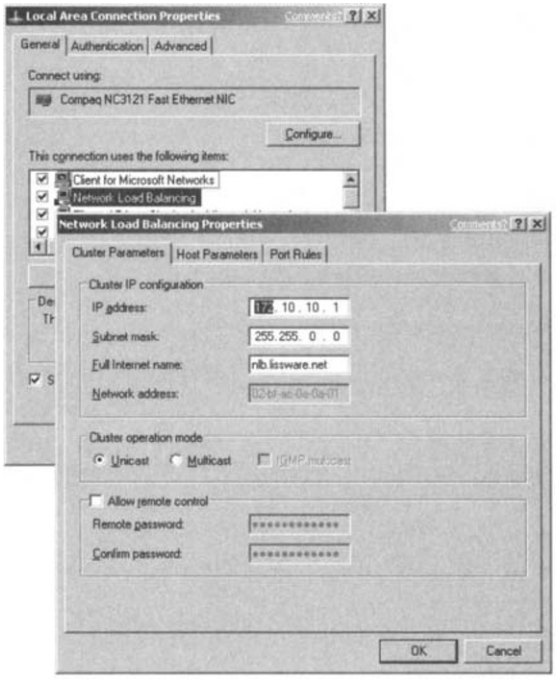

Figure 5.1 *The NLB network adapter user interface.*

The *NlbsNicProv* provider only supports the *NlbsNic* class, which exposes a series of static methods (*Static* Qualifier set to true) to manage the NLB adapter configuration. Basically, the methods can retrieve and set the configuration parameters available from the user interface in the network settings (see Figure 5.1).

Note that the *NlbsNic* class does not expose any properties. All information must be retrieved and set through the methods listed in Table 5.3.

Sample 5.1 shows how to proceed to retrieve information about the NLB network configuration settings.

Table 5.3 *The NlbsNic Class Static Methods*

Name	Description
ControlCluster	Changes local operational status of cluster or port.
GetClusterConfiguration	Gets extended cluster configuration information.
GetCompatibleAdapterGuids	Gets the list of GUIDs of adapters compatible with NLBand the number of adapters currently bound to NLB.
QueryConfigurationUpdateStatus	Queries status of a pending asynchronous cluster configuration update.
RegisterManagementApplication	Registers a management application with NLB.
UnregisterManagementApplication	Unregisters a management application with NLB.
UpdateClusterConfiguration	Asynchronously updates cluster configuration.

Sample 5.1 *Retrieving NLB network configuration settings*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
..:
17:    </runtime>
18:
19:    <script language="VBScript" src="..\Functions\TinyErrorHandler.vbs" />
20:
21:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
22:
23:    <script language="VBscript">
24:    <![CDATA[
..:
28:    Const cComputerName = "LocalHost"
29:    Const cWMINameSpace = "root\MicrosoftNLB"
..:
62:    ' ---------------------------------------------------------------------------
63:    ' Parse the command line parameters
64:    strUserID = WScript.Arguments.Named("User")
65:    If Len(strUserID) = 0 Then strUserID = ""
..:
80:    Set objWMIClass = objWMIServices.Get ("NlbsNIC")
81:    If Err.Number Then ErrorHandler (Err)
82:
83:    intRC = objWMIClass.GetCompatibleAdapterGuids(arrayAdapterGuids, intNumBoundToNlb)
84:    If intRC Then
85:       WScript.Echo "Error getting Compatible Adapter GUID. (" & intRC & ")"
86:       WScript.Quit (1)
87:    End If
88:
89:    For intIndice = 0 To intNumBoundToNlb - 1
90:        WScript.Echo "Adapter GUID #" & intIndice + 1 & ": ................. " & _
91:                     arrayAdapterGuids (intIndice)
92:
93:        intRC = objWMIClass.GetClusterConfiguration (arrayAdapterGuids (intIndice), _
94:                                                     strFriendlyName, _
95:                                                     intGeneration, _
96:                                                     arrayNetworkAddresses, _
97:                                                     boolNLBBound, _
98:                                                     strClusterNetworkAddress, _
99:                                                     strClusterName, _
100:                                                    strTrafficMode, _
101:                                                    arrayPortRules, _
102:                                                    intHostPriority, _
103:                                                    strDedicatedNetworkAddress, _
104:                                                    intClusterModeOnStart, _
105:                                                    boolPersistSuspendOnReboot, _
106:                                                    boolRemoteControlEnabled, _
107:                                                    intHashedRemoteControlPassword)
108:        If intRC Then
109:           WScript.Echo "Error getting Cluster Configuration. (" & intRC & ")"
110:           WScript.Quit (1)
111:        End If
112:
113:        WScript.Echo "Adapter friendly name: ........... " & strFriendlyName
114:        WScript.Echo "Generation: ...................... " & intGeneration
115:        For Each varTemp In arrayNetworkAddresses
```

```
116:            WScript.Echo "Network address(es): ..... " & varTemp
117:        Next
118:        WScript.Echo "NLB bound: ..... " & boolNLBBound
119:        WScript.Echo "Cluster network address: ..... " & strClusterNetworkAddress
120:        WScript.Echo "Cluster name: ..... " & strClusterName
121:        WScript.Echo "Traffic mode: ..... " & strTrafficMode
122:        For Each varTemp In arrayPortRules
123:            WScript.Echo "Rules: ..... " & varTemp
124:        Next
125:        WScript.Echo "Host priority: ..... " & intHostPriority
126:        WScript.Echo "Dedicated network address: ..... " & strDedicatedNetworkAddress
127:        WScript.Echo "Cluster mode on startup: ..... " & intClusterModeOnStart
128:        WScript.Echo "Persist suspend on reboot: ..... " & boolPersistSuspendOnReboot
129:        WScript.Echo "Remote control enabled: ..... " & boolRemoteControlEnabled
130:        WScript.Echo "Hashed remote control password: .. " & intHashedRemoteControlPassword
131:    Next
...:
137:   ]]>
138:   </script>
139: </job>
140:</package>
```

To retrieve NLB network settings, the first thing to do is to retrieve the adapter GUID (lines 83 through 87). This can be done with the *GetCompatibleAdapterGuids* static method exposed by the *NlbsNIC* class (line 83). Once the adapter GUID is known, it can be used to retrieve the cluster configuration with the *GetClusterConfiguration* static method. This method returns the information in a set of variables passed in the method parameters (lines 93 through 107). Once the *GetClusterConfiguration* static method is successfully invoked, the script displays the content of the variables with respect to their types (lines 113 through 130). The execution of Sample 5.1 shows the following output:

```
 1:  C:\>ViewNLBSSettings.wsf
 2:  Microsoft (R) Windows Script Host Version 5.6
 3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:  Adapter GUID #1: ..... {8C4CECDF-1D96-4E2E-9B0C-066BFA705E0A}
 6:  Adapter friendly name: ..... Local Area Connection
 7:  Generation: ..... 1
 8:  Network address(es): ..... 10.10.10.3/255.0.0.0
 9:  NLB bound: ..... True
10:  Cluster network address: ..... 172.10.10.1/255.255.0.0
11:  Cluster name: ..... nlb.lissware.net
12:  Traffic mode: ..... UNICAST
13:  Rules: ..... ip=255.255.255.255 protocol=BOTH start=0 end=65535
                                   mode=MULTIPLE affinity=SINGLE
14:  Host priority: ..... 1
15:  Dedicated network address: ..... 10.10.10.3/255.0.0.0
16:  Cluster mode on startup: ..... 1
17:  Persist suspend on reboot: ..... False
18:  Remote control enabled: ..... False
19:  Hashed remote control password: .. 0
```

We recognize some of the information shown in Figure 5.1, such as the cluster IP address (line 10) or the cluster traffic mode (line 12).

The *NlbsNIC* class exposes many other methods (see Table 5.3). For instance, the *ControlCluster* method can be used to control the cluster state, while the *UpdateClusterConfiguration* method can be used to update the cluster configuration settings. Because this method asynchronously updates the configuration settings (*note:* do not confuse with the asynchronous scripting techniques), it is possible to get the update status by invoking the *QueryConfigurationUpdateStatus* method. This summarizes the capabilities of the *NblsNIC* class supported by the *NlbsNICProv* provider. As usual, **WMI CIM Studio** or the **LoadCIMinXL.Wsf** script (see Sample 4.32 in the appendix) can be used to retrieve information about these classes with their methods.

The *Microsoft|NLB_Provider|V1.0* provider supports all the other classes listed in Table 5.2. These classes represent the node and cluster configuration settings as well but in an object model typical to the CIM repository class representation (instances, associations, etc.). For instance, since a cluster is always made of nodes, and since nodes and clusters have some specific configuration settings, the object model implemented in the CIM repository is made up of associations.

If we start looking from the *MicrosoftNLB_Node* class, we can see that this class is associated with (see Figure 5.2):

- The *MicrosoftNLB_NodeSetting* class via *MicrosoftNLB_NodeNodeSetting* association class
- The *MicrosoftNLB_Cluster* class with *MicrosoftNLB_ParticipatingNode* association class

As part of the node settings, we also have the port rules defining the IP filtering rules. Therefore, the *MicrosoftNLB_NodeSetting* class is associated

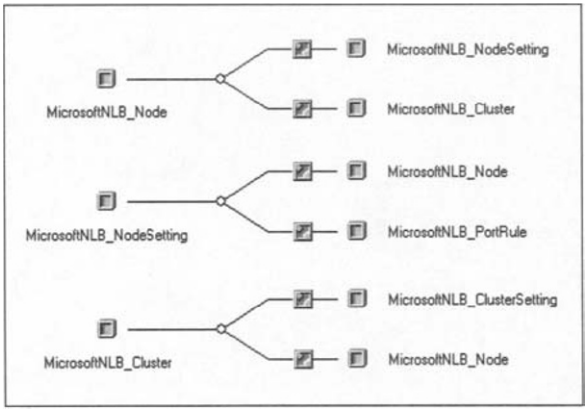

Figure 5.2 *The NLB class associations.*

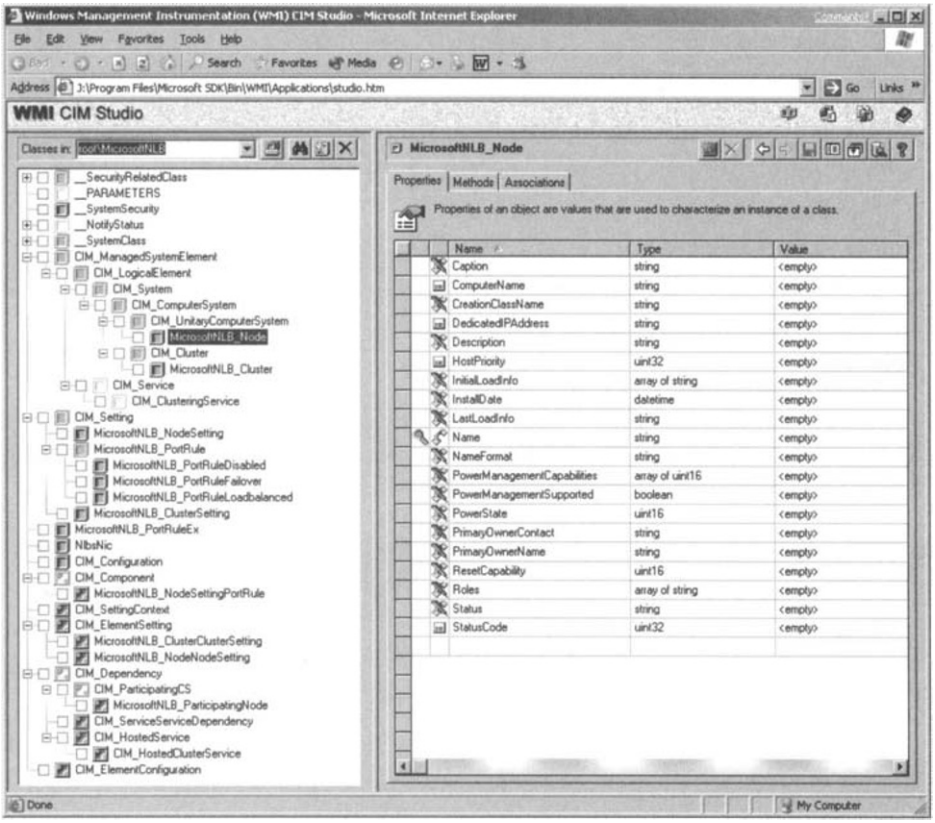

Figure 5.3 *The Node and Cluster classes.*

with the *MicrosoftNLB_PortRule* class, which has three subclasses: *MicrosoftNLB_PortRuleDisabled*, *MicrosoftNLB_PortRuleFailover*, and *MicrosoftNLB_PortRuleLoadbalanced*. In the same way, the *MicrosoftNLB_Cluster* is associated with the *MicrosoftNLB_ClusterSetting* class. This object model is shown with **WMI CIM Studio** (see Figure 5.3).

It is important to note that since no event provider is implemented, this means that any WMI event subscriptions require the **WITHIN** statement in the WQL event query.

5.2.2 Cluster service

Registered in the **Root\MSCluster** namespace, the WMI Cluster support is implemented with three WMI providers, as shown in Table 5.4. These providers are available under Windows Server 2003 only.

Table 5.4 *The Cluster Providers Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
Cluster Providers																
Cluster Event Provider	Root/MSCluster					X						X				
MS_CLUSTER_CLASS_PROVIDER	Root/MSCluster	X						X		X		X				
MS_CLUSTER_PROVIDER	Root/MSCluster		X	X				X	X	X	X	X				

In total more than 30 classes are supported by these providers. The supported classes are listed in Table 5.5.

An interesting point concerns the presence of the *Cluster* WMI event provider. As shown in Table 5.5, this provider supports seven extrinsic event classes, which can be used to track any cluster state modifications. All extrinsic event classes are derived from the *MSCluster_Event* superclass. Therefore, a WQL query such as

```
Select * From MSCluster_Event
```

will detect all changes supported by the event provider and related to the cluster.

Among the 30 classes in Table 5.5, the most interesting ones from a management perspective are the *MSCluster_Node*, *MSCluster_Cluster*, *MSCluster_Service*, *MSCluster_ResourceGroup*, and *MSCluster_Resource*. For instance, with the *MSCluster_Node* class, it is possible to enable the cluster node event log replication at the node level by configuring the *EnableEventLogReplication* property. With the *MSCluster_Cluster SetQuorumResource* method, it is possible to define the quorum resources and customize the Admin extension resources. The *MSCluster_Service AddNode* and *EvictNode* methods allow the addition and the removal of cluster nodes. This class also allows the start and stop of the cluster service, since it is a class derived from the *CIM_Service* superclass. And, last but not least, with the *MSCluster_ResourceGroup* and *MSCluster_Resource* classes, it is possible to bring a cluster resource on-line or off-line. It is also possible to move a resource to another cluster node. As usual, by using **WMI CIM Studio** or the **LoadCIMinXL.Wsf** script, you can gather more information about the class properties and the methods they expose.

Table 5.5 *The Cluster Providers Classes*

Name	Type	Description
MSCluster_Cluster	Dynamic	Represents a cluster.
MSCluster_Network	Dynamic	Represents cluster networks, which define a network as a connection between network interfaces on the same subnet.
MSCluster_NetworkInterface	Dynamic	Represents the network interface used by the cluster.
MSCluster_Node	Dynamic	Represents a cluster node.
MSCluster_Resource	Dynamic	Represents a cluster resource.
MSCluster_ResourceGroup	Dynamic	Represents a cluster group.
MSCluster_ResourceType	Dynamic	Represents a resource type.
MSCluster_Service	Dynamic	Represents a Cluster service is a Windows NT/Windows 2000 component used to control server cluster activities on a single node.
MSCluster_ClusterToNetwork	Association	Represents the networks the cluster uses for communication.
MSCluster_ClusterToNetworkInterface	Association	Represents the network interfaces the cluster has installed on the nodes it manages.
MSCluster_ClusterToNode	Association	Association class that provides access to the nodes in a cluster.
MSCluster_ClusterToQuorumResource	Association	Represents the cluster quorum resource.
MSCluster_ClusterToResource	Association	Represents the resources in a cluster.
MSCluster_ClusterToResourceGroup	Association	Provides access to the groups in a cluster.
MSCluster_ClusterToResourceType	Association	Represents the groups in the cluster.
MSCluster_NetworkToNetworkInterface	Association	Represents the network interfaces connected to a network.
MSCluster_NodeToActiveGroup	Association	Represents the groups active on a node.
MSCluster_NodeToActiveResource	Association	Represents the resources active on a node.
MSCluster_NodeToHostedService	Association	Represents a service managed by the cluster as a resource.
MSCluster_NodeToNetworkInterface	Association	Represents the network interfaces connected to a node.
MSCluster_ResourceGroupToPreferredNode	Association	Represents a list of the resource groups and their preferred nodes list.
MSCluster_ResourceGroupToResource	Association	Represents the resources in a group.
MSCluster_ResourceToDependentResource	Association	Represents the dependencies of a resource.
MSCluster_ResourceToPossibleOwner	Association	Represents a list of the resources and their possible owner nodes.
MSCluster_ResourceTypeToResource	Association	Represents resources of a particular type.
MSCluster_Event	Extrinsic Event	Represents a cluster event.
MSCluster_EventGroupStateChange	Extrinsic Event	Represents a group state change event.
MSCluster_EventObjectAdd	Extrinsic Event	Represents an add object event. An add object event is generated when a cluster object is added to the cluster.
MSCluster_EventObjectRemove	Extrinsic Event	Represents a remove object event. A remove object event is generated when a cluster object is removed from the cluster.
MSCluster_EventPropertyChange	Extrinsic Event	Represents a property change event. A property change event is generated when a cluster object property is changed.
MSCluster_EventResourceStateChange	Extrinsic Event	Represents a resource state change event.
MSCluster_EventStateChange	Extrinsic Event	Represents a state change event. A state change event is generated when the state of a cluster changes.

Because a cluster consists of various components, the WMI classes are linked together with a collection of association classes. For instance, Figure 5.4 illustrates the associations in place for the *MSCluster_Cluster* class.

5.2.3 Terminal Server service

To support the WMI management of Terminal Services under Windows Server 2003, several providers are registered in the **Root\CIMv2** namespace. Basically, there is one WMI provider per management function supported from WMI. This huge number of providers makes the situation

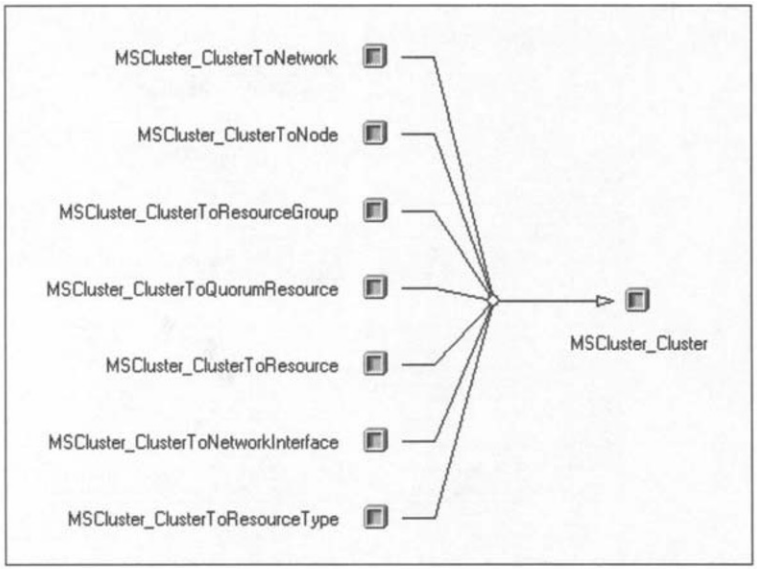

Figure 5.4 *The MSCluster_Cluster class associations.*

a little bit confusing compared with the two previous Windows services. Therefore, we will examine this service in more detail.

Each provider supports one class from Table 5.6. You can determine the provider that supports a selected class by simply looking at the provider

Table 5.6 *The Terminal Server Providers Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
Terminal Server Providers																
Win32_WIN32_TERMINAL_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TERMINALSERVICE_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TERMINALSERVICESETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TERMINALSERVICETOSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TERMINALTERMINALSETTING_Prov	Root/CIMV2		X					X	X	X	X	X	X			
Win32_WIN32_TSACCOUNT_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSCLIENTSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSENVIRONMENTSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSGENERALSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSLOGONSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSNETWORKADAPTERLISTSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSNETWORKADAPTERSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSPERMISSIONSSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSREMOTECONTROLSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSSESSIONDIRECTORY_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSSESSIONDIRECTORYSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_TSSESSIONSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			

name, since the provider name contains the class name. For instance, the *Win32_WIN32_TERMINALSERVICETOSETTING_Prov* provider supports the *Win32_TerminalServiceToSetting* class. Of course, this is a peculiarity of the *Terminal Services* providers naming convention. Previously, to determine which supported a selected class, you would check the *Provider* Qualifier of the class. Even if there is one provider per class, this doesn't change anything from a scripting point of view. This WMI model is mainly designed to customize the Terminal Services configuration settings. The only interesting point to note is that none of the providers is implemented as an event provider, which implies the use of the **WITHIN** statement when monitoring Terminal Services settings. The supported classes are shown in Table 5.7.

With these classes, you can manage the settings available from the "Terminal Services Configuration" MMC. With the *Win32_TerminalServiceSetting* class, you can manage the settings shown in the "Server Settings" folder

Table 5.7 *The Terminal Server Providers Classes*

Name	Type	Description
Win32_Terminal	Dynamic	The Win32_Terminal class is the element of the TerminalSetting association where groups such as: General, Logon, Session, Environment, Remote Control, Client, Network Adapter, and Permission are several configuration setting classes.
Win32_TerminalService	Dynamic	The Win32_TerminalService class provides Terminal Service load-balancing indicators.
Win32_TerminalServiceSetting	Dynamic	The Win32_TerminalServiceSetting class defines the configuration for TerminalServerSetting. This includes capabilities such as Terminal Server Mode, Licensing, Active Desktop, Permissions Capability,Deletion of Temporary folders, and Temporary folders per session.
Win32_TSAccount	Dynamic	The Win32_TSAccount class allows deleting an existing account on the Win32_Terminal class and Modify exisiting Permissions.
Win32_TSClientSetting	Dynamic	The Win32_TSClientSetting class defines the configuration for Win32_Terminal. This includes capabilities such as Connection policy, printer, drive, clipboard mappings, color depth, and connection settings.
Win32_TSEnvironmentSetting	Dynamic	The Win32_TSEnvironmentSetting class defines the configuration for Win32_Terminal. This includes capabilities such as Initial program policy.
Win32_TSGeneralSetting	Dynamic	The Win32_TSGeneralSetting includes capabilities such as Protocol, Transport, Comment, Windows authentication, and Encryption Level.
Win32_TSLogonSetting	Dynamic	The Win32_TSLogonSetting class allows configuring logon settings such as Username, Domain, and Password.
Win32_TSNetworkAdapterListSetting	Dynamic	The Win32_TSNetworkAdapterListSetting class enumerates IP address, GUID, and Name of the Network Adapter.
Win32_TSNetworkAdapterSetting	Dynamic	The Win32_TSNetworkAdapterSetting class defines the configuration for Win32_Terminal. This includes capabilities such as Network Adapter, Maximum Connections, etc.
Win32_TSPermissionsSetting	Dynamic	The Win32_TSPermissionsSetting class allows granting permissions to new accounts and restoring default permissions on the terminal.
Win32_TSRemoteControlSetting	Dynamic	The Win32_TSRemoteControlSetting class defines the configuration for Win32_Terminal. This includes capabilities such as Remote Control policy.
Win32_TSSessionDirectory	Dynamic	The Win32_TSSessionDirectory class defines the configuration for Win32_TSSessionDirectorySetting. This includes capabilities such as Session Directory store, Cluster Name, and Server IP address.
Win32_TSSessionSetting	Dynamic	The Win32_TSSessionSetting class defines the configuration for Win32_Terminal. This includes capabilities such as Time-limits, Disconnection, and Reconnection actions.
Win32_TerminalServiceToSetting	Association	The Win32_TerminalServiceToSetting class is an association with Win32_TerminalService as the Element and Win32_TerminalServiceSetting as the Setting property that includes Terminal Server Mode,Licensing, Active Desktop, Permissions Capability, Deletion of Temporary folders, and Temporary folders per session.
Win32_TerminalTerminalSetting	Association	The Win32_TerminalTerminalSetting class represents an association between a Terminal and its configuration settings.
Win32_TSSessionDirectorySetting	Association	The Win32_TSSessionDirectorySetting class is an association with Win32_TerminalService as the Element and Win32_TSSessionDirectory as the Setting property that includes Terminal Server Session Directory Location, Cluster Name, and SessionDirectoryActive properties.

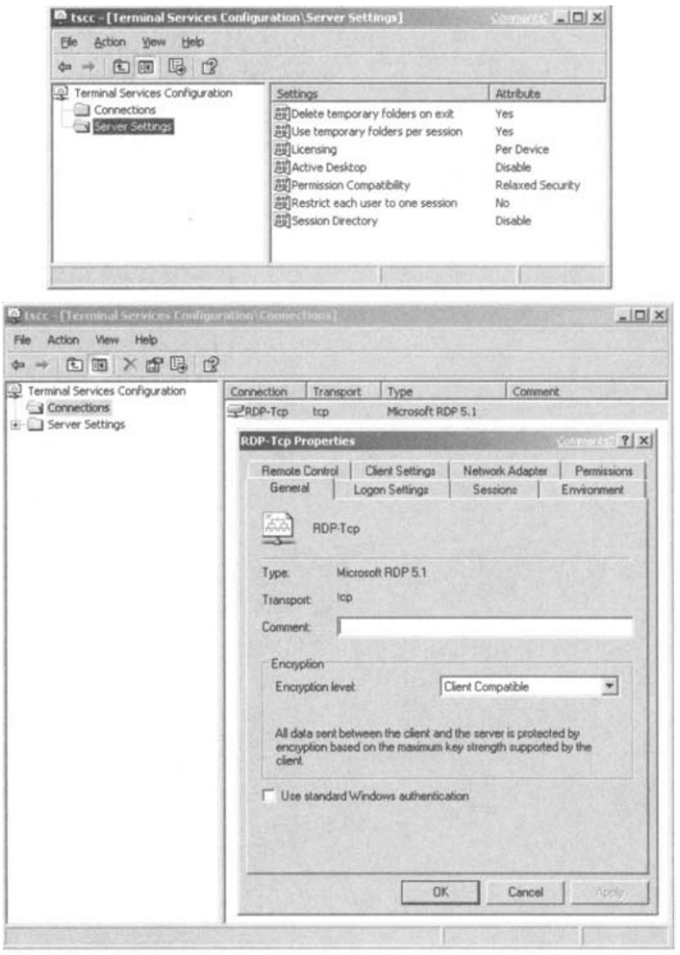

Figure 5.5
The Terminal Services Configuration MMC.

(see Figure 5.5, Server Settings view—top). With all other classes you can manage the settings of the connection transport visible in the "Connections" folder (see Figure 5.5, Connections view—bottom).

5.2.3.1 The Terminal Server configuration

To configure the Terminal Server settings, the *Win32_TerminalServiceSetting* class exposes several methods. Each method corresponds to a setting of the "Server Settings."

As shown in the Sample 5.2 code snippet, the license mode can be changed with the *ChangeMode* method (line 267) and the corresponding value shown in Table 5.8.

Sample 5.2 *Changing the license mode*

```
...:
...:
...:
264:
265:          ' License Mode ------------------------------------------------------------------------
266:          If boolChangeLicenseMode Then
267:             intRC = objWMIInstance.ChangeMode (intChangeLicenseMode)
...:
270:             If intRC = 0 Then
271:                WScript.Echo "TS license mode settings configured."
272:             Else
273:                WScript.Echo "Failed to configure license mode settings (" & intRC & ")."
274:             End If
275:          End If
276:
...:
...:
...:
```

The same logic applies to change the other server settings shown in Figure 5.5, Server Settings view. However, a specific method must be used for each. Therefore the *Win32_TerminalServiceSetting* class exposes the *SetAllowTSConnections*, *SetHomeDirectory*, *SetProfilePath*, *SetSingleSession*, and *SetTimeZoneRedirection* methods. The corresponding values listed in Table

Table 5.8 *The Win32_TerminalServiceSetting Customization Values*

License Mode	Value	Win32_TerminalServiceSetting Method/Property
Admin	1	*ChangeMode*() method
PerDevice	2	
PerSession	3	
Personal	4	
Security Mode		
Full	0	*UserPermission* property
Relaxed	1	
TS Connections		
Allow	1	*SetAllowTSConnections*() method
Deny	0	
Home Directory		
	C:\MyHomePath	*SetHomeDirectory*() method
Profile Path		
	C:\MyProfilePath	*SetProfilePath*() method
Delete Temp Folders		
Enabled	1	*SetPolicyPropertyName*() method
Disabled	0	
Use Temp Folders		
Enabled	1	*SetPolicyPropertyName*() method
Disabled	0	
Single Session		
Enabled	1	*SetSingleSession*() method
Disabled	0	
Time Zone Redirection		
Enabled	1	*SetTimeZoneRedirection*() method
Disabled	0	

5.8 must be used as the method parameter. In some cases, to customize the *Win32_TerminalServiceSetting* instance, it is necessary to update a property. This is the case for the security mode. In such a case, the script must set the *UserPermission* property (see Sample 5.3, lines 279 through 281) with the corresponding value (see Table 5.8).

Sample 5.3 *Changing the security mode*

```
...:
...:
...:
276:
277:          ' Security Mode ------------------------------------------------------------------
278:          If boolSecurityMode Then
279:             objWMIInstance.UserPermission = intSecurityMode
280:             objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
281:                                  wbemFlagReturnWhenComplete)
...:
284:             WScript.Echo "TS security mode configured."
285:          End If
286:
...:
...:
...:
```

A final peculiarity concerning the *Win32_TerminalServiceSetting* class is the modification of the temporary folder settings (see Sample 5.4). These settings must be set with the *SetPolicyPropertyName* method (lines 325 and 326, lines 338 and 339), which accepts two parameters: The first parameter corresponds to the property name to configure, and the second parameter corresponds to the value assigned to the selected property name.

Sample 5.4 *Changing the temporary folder settings*

```
...:
...:
...:
322:
323:          ' Delete Temp Folders ------------------------------------------------------------
324:          If boolDeleteTempFolders Then
325:             intRC = objWMIInstance.SetPolicyPropertyName ("DeleteTempFolders", _
326:                                                           intDeleteTempFolders)
...:
329:             If intRC = 0 Then
330:                WScript.Echo "TS Delete Temp Folders setting configured."
331:             Else
332:                WScript.Echo "Failed to configure Delete Temp Folders setting(" & intRC & ")."
333:             End If
334:          End If
335:
336:          ' Use Temp Folders ----------------------------------------------------------------
337:          If boolUseTempFolders Then
338:             intRC = objWMIInstance.SetPolicyPropertyName ("UseTempFolders", _
339:                                                           intUseTempFolders)
...:
```

```
342:             If intRC = 0 Then
343:                WScript.Echo "TS Use Temp Folders setting configured."
344:             Else
345:                WScript.Echo "Failed to configure Use Temp Folders setting(" & intRC & ")."
346:             End If
347:          End If
348:
...:
...:
...:
```

5.2.3.2 The Terminal Server connections configuration

To manage the Terminal Services connections settings, all other classes from Table 5.7 must be used. Even if the overall logic is the same as for the "Server Settings" with the *Win32_TerminalServiceSetting* class, the Terminal Services connections settings are a bit more complex to set up due to the number of configuration possibilities. In any case, the first action is to retrieve the connection settings instance. By default the "RDP-tcp" connection is the only connection setting instance available (see Figure 5.5, Connections view).

5.2.3.2.1 Enabling/disabling the Terminal Server connections

Enabling or disabling the connection settings is the easiest configuration to perform, since it involves the invocation of the *Enable* method of the *Win32_Terminal* class (see Sample 5.5). A parameter value of 1 enables the connection settings instance, while a parameter value of 0 disables it (line 594). This action corresponds to a right click on the connection name available in the MMC and selecting "Enable connection" or "Disable connection" (in the "All tasks" submenu).

Sample 5.5 *Enabling/disabling the Terminal Services connections*

```
...:
...:
...:
587:
588:    ' Enable/Disable ----------------------------------------------------------------------------
589:    If boolTerminalEnable Then
590:       Set objWMIInstance = objWMIServices.Get ("Win32_Terminal='" & _
591:                                                strTerminalName & "'")
592:       If Err.Number Then ErrorHandler (Err)
593:
594:       intRC = objWMIInstance.Enable (intTerminalEnable)
...:
597:       If intRC = 0 Then
598:          If intTerminalEnable = 1 Then
599:             WScript.Echo "TS enabled."
600:          Else
601:             WScript.Echo "TS disabled."
602:          End If
```

```
603:       Else
604:          WScript.Echo "Failed to enable/disable Terminal Server (" & intRC & ")."
605:       End If
606:    End If
607:
...:
...:
...:
```

5.2.3.2.2 The Terminal Services remote control settings

To configure the remote control connection settings, the *RemoteControl* method exposed by the *Win32_TSRemoteControlSetting* class must be invoked. To properly customize all combinations available from the user interface (see Figure 5.6), it is necessary to set up the *RemoteControlPolicy* property before any other settings.

To set up the various combinations, the two parameters (the *RemoteControl* method parameter and the *RemoteControlPolicy* property) must combine different values. Table 5.9 shows the various combinations for both values with their corresponding settings.

Sample 5.6 shows the coding logic to update the *RemoteControlPolicy* property (lines 614 through 616) and invoke the *RemoteControl* method (lines 619 and 620) if the policy is enabled.

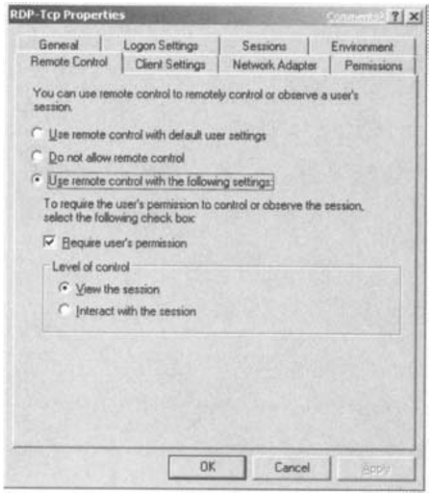

Figure 5.6 *The Terminal Services remote control configuration.*

Table 5.9 *The Terminal Services Remote Control Configuration Values*

	LevelOfControl	RemoteControlPolicy
Use remote control with default user settings	0	1
Do not allow remote control	0	0
Use remote control with the following settings		
Require user's permission is **checked**		
View the session	3	0
Interact with the session	1	0
Require user's permission is **unchecked**		
View the session	4	0
Interact with the session	2	0

Sample 5.6 *Configuring the Terminal Services remote control settings*

```
...:
...:
...:
607:
608:    ' Remote Control ------------------------------------------------------------------------
609:    If boolRemoteControl Then
610:       Set objWMIInstance = objWMIServices.Get ("Win32_TSRemoteControlSetting='" & _
611:                                                strTerminalName & "'")
...:
614:       objWMIInstance.RemoteControlPolicy = intRemoteControlPolicy
615:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
616:                            wbemFlagReturnWhenComplete)
...:
619:       If intRemoteControlPolicy = 0 Then
620:          intRC = objWMIInstance.RemoteControl (intLevelOfControl)
...:
622:       End If
623:
624:       If intRC = 0 Then
625:          WScript.Echo "TS Remote Control policy configured."
626:       Else
627:          WScript.Echo "Failed to configure the Remote Control policy (" & intRC & ")."
628:       End If
629:    End If
630:
...:
...:
...:
```

5.2.3.2.3 The Terminal Services network adapter and connection limit settings

To set up the connection limit, it is necessary to update the *MaximumConnections* property exposed by the *Win32_TSNetworkAdapterSetting* instance representing the Terminal Services network adapter and connection limit settings (see Figure 5.7).

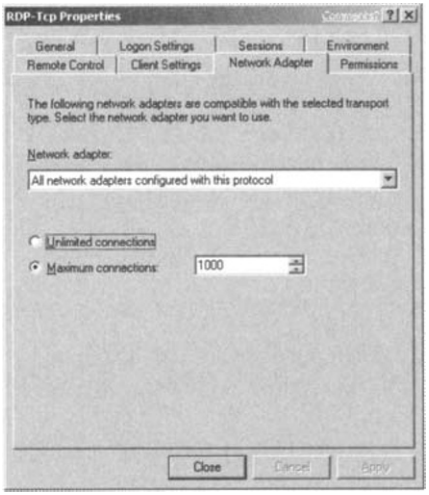

Figure 5.7 *The Terminal Service connection limit settings.*

A value of 4,294,967,295 (which is 2^32 – 1, the biggest unsigned integer on 32 bits) sets the connection limit to unlimited. Any other value determines the connection limit in seconds. Sample 5.7 illustrates the coding logic (lines 637 through 639).

Sample 5.7 *Configuring the Terminal Services maximum connection settings*

```
...:
...:
...:
630:
631:    ' Maximum TS connections ------------------------------------------------------------------
632:    If boolMaximumConnections Then
633:       Set objWMIInstance = objWMIServices.Get ("Win32_TSNetworkAdapterSetting='" & _
634:                                                strTerminalName & "'")
...:
637:       objWMIInstance.MaximumConnections = longMaximumConnections
638:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
639:                            wbemFlagReturnWhenComplete)
...:
642:       If longMaximumConnections = cNoConnectionLimit Then
643:          WScript.Echo "TS has no connections limit configured."
644:       Else
645:          WScript.Echo "TS has " & longMaximumConnections & " connections limit configured."
646:       End If
647:    End If
648:
649:    ' Adapter -----------------------------------------------------------------------------------
650:    If boolAdapter Then
651:       Set objWMIInstance = objWMIServices.Get ("Win32_TSNetworkAdapterSetting='" & _
652:                                                strTerminalName & "'")
...:
655:       If UCase (strIPAddress) = "ALL" Then
656:          intRC = objWMIInstance.SelectAllNetworkAdapters
```

```
...:
658:        Else
659:           intRC = objWMIInstance.SelectNetworkAdapterIP (strIPAddress)
...:
661:        End If
662:
663:        If intRC = 0 Then
664:           WScript.Echo "TS adapter configured."
665:        End If
666:     End If
667:
...:
...:
...:
```

From the *Win32_TSNetworkAdapterSetting* instance it is also possible to determine which adapter is compatible with the selected Terminal Services Transport. To set up all adapters, the *SelectAllNetworkAdapters* method must be invoked (line 656). To select a specific network adapter, the *SelectNetworkAdapterIP* method must be invoked with the adapter IP address as a parameter (line 659).

5.2.3.2.4 The Terminal Services encryption and authentication settings

To configure the encryption level, the *SetEncryptionLevel* method exposed by the *Win32_TSGeneralSetting* class must be invoked. To configure the authentication method, the *WindowsAuthentication* property exposed by the *Win32_TSGeneralSetting* instance must be updated.

Figure 5.8 shows the various settings available, while Table 5.10 contains the miscellaneous values.

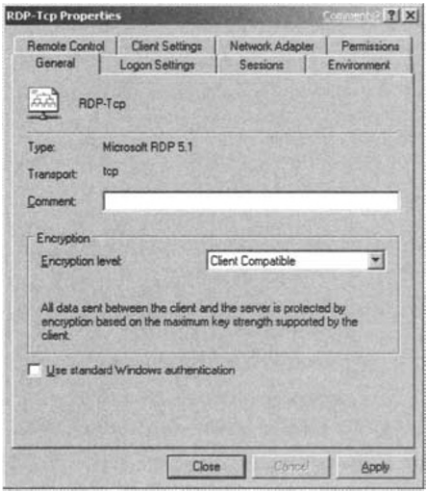

Figure 5.8 *The Terminal Services general connection settings.*

Table 5.10 *The Terminal Services Encryption and Authentication Level Values*

	MinEncryptionLevel	WindowsAuthentication
High	3	
Client Compatible	2	
Use standard Windows Authentication **checked**		1
Use standard Windows Authentication **unchecked**		0

Sample 5.8 illustrates the coding logic.

Sample 5.8 *Configuring the Terminal Services encryption and authentication levels*

```
...:
...:
...:
667:
668:    ' Encryption level ------------------------------------------------------------------------
669:    If boolEncryptionLevel Then
670:       Set objWMIInstance = objWMIServices.Get ("Win32_TSGeneralSetting='" & _
671:                                                strTerminalName & "'")
...:
674:       intRC = objWMIInstance.SetEncryptionLevel(intEncryptionLevel)
...:
677:       WScript.Echo "TS encryption level configured."
678:    End If
679:
680:    ' Windows Authentication -------------------------------------------------------------------
681:    If boolWinAuthentication Then
682:       Set objWMIInstance = objWMIServices.Get ("Win32_TSGeneralSetting='" & _
683:                                                strTerminalName & "'")
...:
686:       objWMIInstance.WindowsAuthentication = intWinAuthentication
687:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
688:                            wbemFlagReturnWhenComplete)
...:
691:       If intRC = 0 Then
692:          WScript.Echo "TS Windows authentication configured."
693:       End If
694:    End If
695:
...:
...:
...:
```

5.2.3.2.5 The Terminal Services client settings

To set up the Terminal Services default client settings, various methods and properties exposed by the *Win32_TSClientSetting* class must be used. Figure 5.9 shows the settings available.

The number of values that could be set to customize the Terminal Services default client settings is quite confusing, because it is sometimes necessary to update a property directly by assigning a new value; at other times, it may be necessary to invoke a method instead (see Table 5.11).

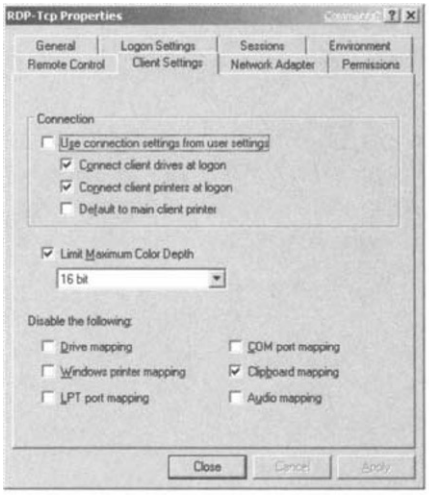

Figure 5.9 *The Terminal Services default client settings.*

Table 5.11 *The Terminal Services Default Client Settings Values*

	ConnectionPolicy
Use connection settings from user settings **unchecked**	0
Use connection settings from user settings **checked**	1
	ConnectClientDrivesAtLogon
Connect client drives at logon **unchecked**	0
Connect client drives at logon **checked**	1
	ConnectPrinterAtLogon
Connect client printers at logon **unchecked**	0
Connect client printers at logon **checked**	1
	DefaultToClientPrinter
Default to main client printer **unchecked**	0
Default to main client printer **checked**	1
	ColorDepthPolicy
Limit color depth **unchecked**	1
Limit color depth **checked**	0
	ColorDepth
8-bit	1
15-bit	2
16-bit	3
24-bit	4
	DriveMapping
Drive mapping **unchecked**	0
Drive mapping **checked**	1
	WindowsPrinterMapping
Windows printer mapping **unchecked**	0
Windows printer mapping **checked**	1
	LPTPortMapping
LPT port mapping **unchecked**	0
LPT port mapping **checked**	1
	COMPortMapping
COM port mapping **unchecked**	0
COM port mapping **checked**	1
	ClipboardMapping
Clipboard mapping **unchecked**	0
Clipboard mapping **checked**	1
	AudioMapping
Audio mapping **unchecked**	0
Audio mapping **checked**	1

By updating the *ConnectionPolicy* property exposed by the *Win32_TSClientSetting* class, a script will perform the exact same change as selecting the "Use connection settings from user settings" check box. Sample 5.9 shows the coding logic.

Sample 5.9 *Configuring the Terminal Services client connection policy settings*

```
...:
...:
...:
695:
696:    ' User Connection settings ------------------------------------------------------------------
697:    If boolConnUserSettings Then
698:       Set objWMIInstance = objWMIServices.Get ("Win32_TSClientSetting='" & _
699:                                                strTerminalName & "'")
...:
702:       objWMIInstance.ConnectionPolicy = intConnUserSettings
703:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
704:                            wbemFlagReturnWhenComplete)
...:
707:       If intConnUserSettings = 0 Then
708:          intRC = objWMIInstance.ConnectionSettings (intConnClientDrivesAtLogon, _
709:                                                     intConnClientPrintersAtLogon, _
710:                                                     intDefaultToMainClientPrinter)
...:
712:       End If
713:
714:       If intRC = 0 Then
715:          WScript.Echo "TS user connection settings configured."
716:       Else
717:          WScript.Echo "Failed to configure user connection settings (" & intRC & ")."
718:       End If
719:    End If
720:
...:
...:
...:
```

If the "Use connection settings from user settings" check box is unchecked, a client connection policy must be defined. Therefore, the three check boxes in the "Connection" frame (See Figure 5.9) can be configured. This is the reason why Sample 5.9 invokes the *ConnectionSettings* method exposed by the *Win32_TSClientSetting* class and sets the state of the three check boxes (lines 708 through 710).

Regarding the color depth policy, the overall logic is the same as for the client connection policy. Sample 5.10 shows the logic. To define the color depth policy, the *SetColorDepthPolicy* method must be invoked with a parameter value defining the check box state (line 727). Table 5.11 lists the miscellaneous values to use. If the color depth policy is enabled, the color depth must be set by invoking the *SetColorDepth* method, with a parameter value corresponding to the number of bits defining the color depth (line 731).

Sample 5.10 *Configuring the Terminal Services client color depth policy settings*

```
...:
...:
...:
720:
721:    ' Color depth settings ------------------------------------------------------------------
722:    If boolColorDepth Then
723:       Set objWMIInstance = objWMIServices.Get ("Win32_TSClientSetting='" & _
724:                                                strTerminalName & "'")
...:
727:       intRC = objWMIInstance.SetColorDepthPolicy (intColorDepth)
...:
730:       If intColorDepth = 0 Then
731:          intRC = objWMIInstance.SetColorDepth (intColorDepthBit)
...:
733:       End If
734:
735:       If intRC = 0 Then
736:          WScript.Echo "TS color depth settings configured."
737:       Else
738:          WScript.Echo "Failed to configure color depth settings (" & intRC & ")."
739:       End If
740:    End If
741:
...:
...:
...:
```

The final settings concerning the client connection settings concern the resource mappings. These settings correspond to the drive, Windows printer, LPT port, COM port, clipboard, and audio mapping (see Figure 5.9). Here, the logic is a little bit different, since there is no dedicated property or method to use for each setting. To configure these parameters, the *SetClientProperty* method exposed by the *Win32_TSClientSetting* class must be used. Sample 5.11 shows the logic to use for the LPT port mapping, but the technique is the same for all mappings. Basically, the first method parameter contains the property name and the second parameter a Boolean value defining the state (line 748). You can check the *Win32_TSClientSetting* class properties to determine the mapping properties to use during the method invocation.

Sample 5.11 *Configuring the Terminal Services client mapping settings*

```
...:
...:
...:
741:
742:    ' LPT Resource mapping ------------------------------------------------------------------
743:    If Len (boolLPTMapping) Then
744:       Set objWMIInstance = objWMIServices.Get ("Win32_TSClientSetting='" & _
745:                                                strTerminalName & "'")
...:
```

```
748:       intRC = objWMIInstance.SetClientProperty ("LPTPortMapping", boolLPTMapping)
...:
751:       If intRC = 0 Then
752:          WScript.Echo "TS LPT port mapping configured."
753:       Else
754:          WScript.Echo "Failed to configure LPT port mapping (" & intRC & ")."
755:       End If
756:    End If
757:
...:
...:
...:
```

5.2.3.2.6 The Terminal Services environment settings

To configure the environment settings of the Terminal Services connection, the *Win32_TSEnvironmentSetting* class must be used. The first setting concerns the initial program policy (see Figure 5.10), which allows the startup of a specific application at logon time.

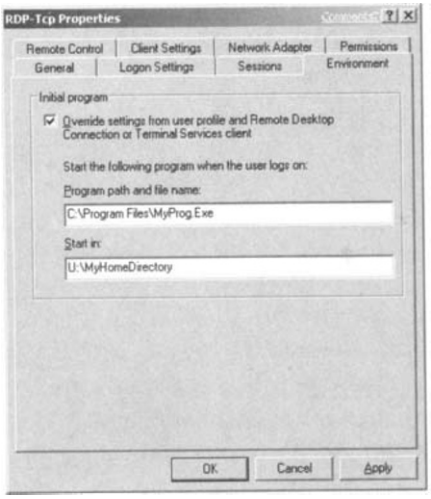

Figure 5.10 *The Terminal Services connection environment settings.*

As shown in Sample 5.12, to define the initial program policy the *InitialProgramPolicy* property must be updated (lines 844 and 845) accordingly to the values listed in Table 5.12. If the policy is enabled, the *InitialProgram* method exposed by the *Win32_TSEnvironmentSetting* class must be invoked (lines 849 through 852), with two parameters containing the program name and the initial program path, respectively.

Sample 5.12 *Configuring the Terminal Services connection environment settings*

```
...:
...:
...:
837:
838:    ' Initial program ---------------------------------------------------------------------------
839:    If boolInitialProgram Then
840:       Set objWMIInstance = objWMIServices.Get ("Win32_TSEnvironmentSetting='" & _
841:                                                strTerminalName & "'")
...:
844:       objWMIInstance.InitialProgramPolicy = intInitialProgramPolicy
845:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
846:                            wbemFlagReturnWhenComplete)
...:
849:       If intInitialProgramPolicy = 0 Then
850:          intRC = objWMIInstance.InitialProgram (strInitialProgramName, strInitialProgramPath)
...:
852:       End If
853:
854:       If intRC = 0 Then
855:          WScript.Echo "TS initial program configured."
856:       Else
857:          WScript.Echo "Failed to configure initial program (" & intRC & ")."
858:       End If
859:    End If
860:
861:    ' Display Wallpaper --------------------------------------------------------------------------
862:    If boolDisplayWallPaper Then
863:       Set objWMIInstance = objWMIServices.Get ("Win32_TSEnvironmentSetting='" & _
864:                                                strTerminalName & "'")
...:
867:       intRC = objWMIInstance.SetClientWallPaper (intDisplayWallPaper)
...:
870:       If intRC = 0 Then
871:          WScript.Echo "TS display wallpaper configured."
872:       Else
873:          WScript.Echo "Failed to configure display wallpaper (" & intRC & ")."
874:       End If
875:    End If
876:
...:
...:
...:
```

Table 5.12 *The Terminal Services Connection Environment Policy Settings*

TS environment settings	InitialProgramPolicy
Override settings from user profile and Remote Desktop Connection or Terminal Services Client **unchecked**	1
Override settings from user profile and Remote Desktop Connection or Terminal Services Client **checked**	0

The wallpaper setting is set with the *SetClientWallPaper* method exposed by the *Win32_TSEnvironmentSetting* class. This method sets the *ClientWallPaper* property exposed by the same class. When the *SetClientWallPaper* method parameter is set to 1, the *ClientWallPaper* property is set to 1 as well and forces the display of the wallpaper on the client desktop.

5.2.3.2.7 The Terminal Services connection logon settings

The *Win32_TSLogonSetting* class configures the Terminal Services connection logon settings shown in Figure 5.11.

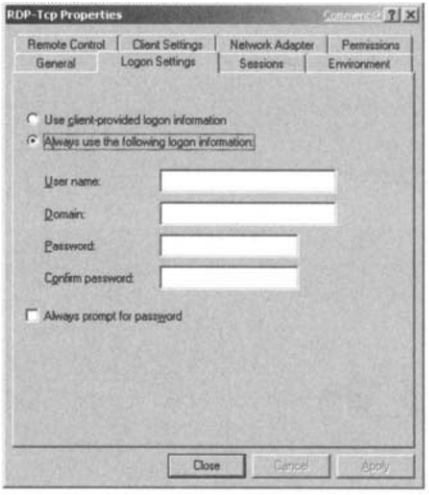

Figure 5.11 *The Terminal Services connection logon settings.*

As shown in Sample 5.13, to configure the client logon policy the *ClientLogonInfoPolicy* property exposed by the *Win32_TSLogonSetting* class must be updated (lines 883 through 885). If the policy is enabled, the logon parameters must be defined with the *ExplicitLogon* method exposed by the *Win32_TSLogonSetting* class (lines 888 through 893). The values to use are listed in Table 5.13.

Table 5.13 *The Terminal Services Connection Logon Values*

	ClientLogonInfoPolicy
Use client-provided logon information	1
Always use the following logon information	0
	PromptForPassword
Always prompt for password **unchecked**	0
Always prompt for password **checked**	1

Sample 5.13 *Configuring the Terminal Services connection logon settings*

```
...:
...:
...:
876:
877:    ' Logon as ------------------------------------------------------------------------------
878:    If boolClientLogon Then
879:       Set objWMIInstance = objWMIServices.Get ("Win32_TSLogonSetting='" & _
880:                                                strTerminalName & "'")
...:
883:       objWMIInstance.ClientLogonInfoPolicy = intClientLogonPolicy
884:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
885:                            wbemFlagReturnWhenComplete)
...:
888:       If intClientLogonPolicy= 0 Then
889:          intRC = objWMIInstance.ExplicitLogon (strLogonUser, _
890:                                                strLogonDomain, _
891:                                                strLogonPassword)
...:
893:       End If
894:
895:       If intRC = 0 Then
896:          WScript.Echo "TS client logon information configured."
897:       Else
898:          WScript.Echo "Failed to configure client logon information (" & intRC & ")."
899:       End If
900:    End If
901:
902:    ' Prompt password ------------------------------------------------------------------------
903:    If boolPromptPassword Then
904:       Set objWMIInstance = objWMIServices.Get ("Win32_TSLogonSetting='" & _
905:                                                strTerminalName & "'")
...:
908:       intRC = objWMIInstance.SetPromptForPassword (intPromptPassword)
...:
911:       If intRC = 0 Then
912:          WScript.Echo "TS prompt password configured."
913:       Else
914:          WScript.Echo "Failed to configure prompt password (" & intRC & ")."
915:       End If
916:    End If
917:
...:
...:
...:
```

To determine if the user must be prompted for the password, the *SetPromptPassword* method exposed by the *Win32_TSLogonSetting* class must be invoked (line 908). A value of 1 will prompt the user for the password.

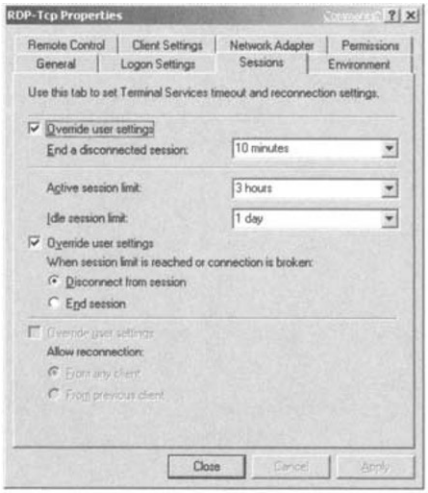

Figure 5.12 *The Terminal Services connection session settings.*

5.2.3.2.8 The Terminal Services connection session settings

The *Win32_TSSessionSetting* class exposes properties to configure the Terminal Services connection session settings shown in Figure 5.12.

The overall logic shown in Sample 5.14 is the same as before. The Terminal Services session policy must be defined first (lines 924 through 926) by updating the *TimeLimitPolicy* property exposed by the *Win32_TSSessionSetting* class. The value to use is shown in Table 5.14. If the policy is enabled, the *TimeLimit* method exposed by the *Win32_TSSessionSetting*

Table 5.14 *The Terminal Services Connection Session Values*

	TimeLimitPolicy
Override user settings **unchecked**	0
Override user settings **checked**	1
	DisconnectedSessionLimit
End a disconnected session	
Never	0
n minutes	Delay in Min * 60 * 1000
	ActiveSessionLimit
Active session limit	
Never	0
n minutes	Delay in Min * 60 * 1000
	IdleSessionLimit
Idle session limit	
Never	0
n minutes	Delay in Min * 60 * 1000
	BrokenConnectionPolicy
Override user settings **unchecked**	1
Override user settings **checked**	0
	BrokenConnectionAction
Disc. From session	0
End session	1
	ReconnectionPolicy
Override user settings **checked**	1

class will define the *DisconnectedSessionLimit* (line 930), *ActiveSessionLimit* (line 937), and *IdleSessionLimit* (line 944) properties. The values to use are listed in Table 5.14.

Sample 5.14 *Configuring the Terminal Services connection session settings*

```
...:
...:
...:
917:
918:    ' User Session settings ------------------------------------------------------------------------
919:    If boolUserTimeLimitSettings Then
920:       Set objWMIInstance = objWMIServices.Get ("Win32_TSSessionSetting='" & _
921:                                                strTerminalName & "'")
...:
924:       objWMIInstance.TimeLimitPolicy = intUserTimeLimitPolicy
925:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
926:                            wbemFlagReturnWhenComplete)
...:
929:       If intUserTimeLimitPolicy = 0 Then
930:          intRC = objWMIInstance.TimeLimit ("DisconnectedSessionLimit", intEndDiscSession)
...:
933:          If intRC Then
934:             WScript.Echo "Failed to configure disconnected session limit (" & intRC & ")."
935:          End If
936:
937:          intRC = objWMIInstance.TimeLimit ("ActiveSessionLimit", intActiveSessionLimit)
...:
940:          If intRC Then
941:             WScript.Echo "Failed to configure Active session limit (" & intRC & ")."
942:          End If
943:
944:          intRC = objWMIInstance.TimeLimit ("IdleSessionLimit", intIdleSessionLimit)
...:
947:          If intRC Then
948:             WScript.Echo "Failed to configure idle session limit (" & intRC & ")."
949:          End If
950:       End If
951:
952:       WScript.Echo "TS user session settings configured."
953:    End If
954:
955:    ' Broken Session ---------------------------------------------------------------------------
956:    If boolBrokenSession Then
957:       Set objWMIInstance = objWMIServices.Get ("Win32_TSSessionSetting='" & _
958:                                                strTerminalName & "'")
...:
961:       objWMIInstance.BrokenConnectionPolicy = intBrokenConnectionPolicy
962:       objWMIInstance.Put_ (wbemChangeFlagCreateOrUpdate Or _
963:                            wbemFlagReturnWhenComplete)
...:
966:       If intBrokenConnectionPolicy = 0 Then
967:          intRC = objWMIInstance.BrokenConnection (intBrokenConnectionAction)
...:
969:       End If
970:
971:       If intRC = 0 Then
```

```
972:          WScript.Echo "TS broken connection action settings configured."
973:       Else
974:          WScript.Echo "Failed to configure broken connection action settings (" & intRC & ")."
975:       End If
976:    End If
977:
...:
...:
...:
```

The same logic applies for the broken session policy. The *BrokenConnectionPolicy* property exposed by the *Win32_TSSessionSetting* class must be set first (lines 961 through 963). If the policy is enabled, the *BrokenConnectionAction* property must be set with the *BrokenConnection* method (line 967).

5.2.3.2.9 The Terminal Services connection permissions settings

As we have seen in Chapter 4, managing the security often implies the manipulation of a security descriptor. However, to modify the Terminal Services connection permissions settings, it is necessary to invoke specific methods exposed by the *Win32_TSPermissionsSetting* class (see Figure 5.13).

The *Win32_TSPermissionsSetting* class exposes the *AddAccount* method, which takes two parameters: the Domain\UserID account name and the permission value, which defines the right granted for the given account name (see Table 5.15, "Values" column).

To restore the default permissions, the *Win32_TSPermissionsSetting* class exposes the *RestoreDefaults* method. To delete an account granted on the

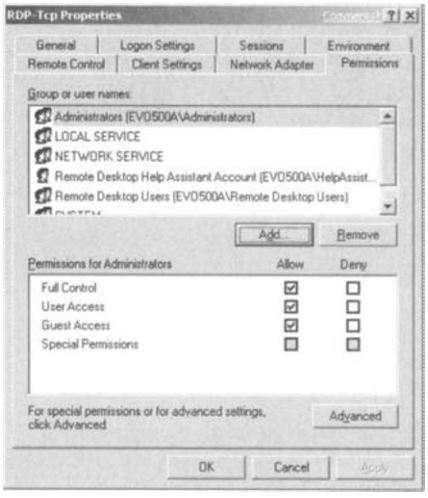

Figure 5.13 *The Terminal Services connection permission settings.*

Table 5.15 *The Terminal Services Permission Values and Masks*

TS permissions settings	Values	Masks
Guest	0	32
User and Guest	1	417
Full control, User and Guest	2	983999

Terminal Services connection settings, the *Delete* method exposed by the *Win32_TSAccount* class must be used. If the granted permissions must be modified, the *Win32_TSAccount* class exposes the *ModifyPermissions* and *ModifyAuditPermissions* methods. Both methods require an access mask (see Table 5.15, "Masks" column) and a Boolean value to determine if the access mask is granted or denied.

5.2.4 Windows Driver Model provider

The *Windows Driver Model* (WDM) provider is available from Windows 2000 and later. This provider gives access to WMI information exposed by drivers that are WDM enabled. WDM is an operating system interface through which hardware components (device drivers) provide information exposed by WMI (see Table 5.16). Actually, the *WDM* provider is made up of two providers registered in the **Root\WMI** namespace:

- A class, instance, and method provider called *WMIProv*
- An event provider called *WMIEventProv*

We will not go into the details of the WDM operating system interface. However, to retrieve information from the WDM-enabled drivers, WDM drivers must expose WMI information loaded in the CIM repository. To

Table 5.16 *The WDM Providers Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
WDM Provider																
WMIProv	Root/WMI	X	X	X				X	X	X		X	X	X	X	
WMIProv	Root/MicrosoftNLB	X	X	X				X	X	X		X				
WMIEventProv	Root/WMI					X						X	X	X		
WMIEventProv	Root/MicrosoftNLB					X						X				

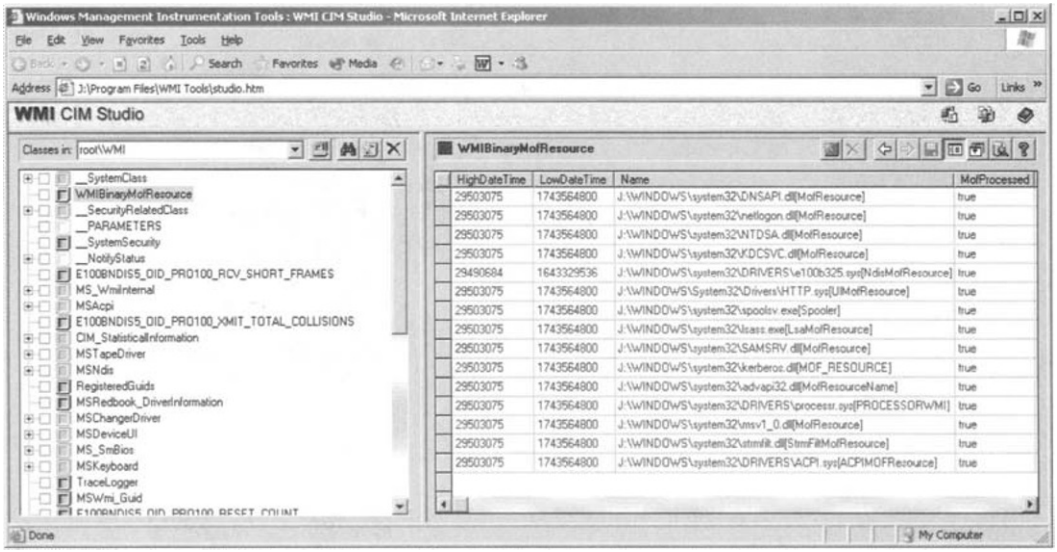

Figure 5.14 *The WMIBinaryMofResource class instances.*

determine the list of drivers exposing information in the CIM repository, it is possible to request instances of the *WMIBinaryMofResource* class (see Figure 5.14).

The most interesting classes supported by the instance provider are summarized in Table 5.17.

Since an event provider is also available, an extrinsic event class can be used in the WQL event queries. The event class is called *WMIEvent* and is used as a parent class to define a collection of extrinsic event classes (see Figure 5.15).

To give a first example of the *WDM* providers classes use, we can reuse the **GetCollectionOfInstances.wsf** script (see Sample 1.5, "Listing all instances of a class with their properties formatted") and request all instances of the *MSDiskDriver_Geometry* class.

```
C:\>GetCollectionOfInstances.wsf MSDiskDriver_Geometry /Namespace:Root\WMI
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.


Active: ................................ TRUE
BytesPerSector: ........................ 512
Cylinders: ............................. 5169
*InstanceName: ......................... IDE\DiskMAXTOR_6L040J2_...__A93.0500\3631... 2020202020_0
MediaType: ............................. 12
SectorsPerTrack: ....................... 63
TracksPerCylinder: ..................... 240
```

Table 5.17 *The WDM Providers Classes*

Name	Description
MSAcpi_ThermalZoneTemperature	ThermalZone temperature information
MSAcpiInfo	ACPI Table data
MSChangerParameters	Changer Parameters
MSChangerProblemDeviceError	Changer Errors
MSChangerProblemEvent	Changer Problem Warning
MSDeviceUI_FirmwareRevision	Firmware Revision
MSDiskDriver_Geometry	Disk Geometry
MSDiskDriver_Performance	Disk performance statistics
MSIde_PortDeviceInfo	Scsi Address
MSKeyboard_ClassInformation	Keyboard class driver information
MSKeyboard_ExtendedID	Keyboard port extended ID
MSKeyboard_PortInformation	Keyboard port driver information
MSMCAEvent_CPUError	MCA CPU Error Event
MSMCAEvent_InvalidError	MCA Unknown Error Event
MSMCAEvent_MemoryError	MCA Memory Error Event
MSMCAEvent_MemoryPageRemoved	Memory page has been removed
MSMCAEvent_PCIBusError	MCA PCI Bus Error Event
MSMCAEvent_PCIComponentError	MCA PCI Platform Component Error Event
MSMCAEvent_PlatformSpecificError	MCA Platform Specific Error Event
MSMCAEvent_SMBIOSError	MCA SMBIOS Error Event
MSMCAEvent_SwitchToCMCPolling	CMC handling switched from interrupt driver to polling
MSMCAEvent_SwitchToCPEPolling	CPE handling switched from interrupt driver to polling
MSMCAEvent_SystemEventError	MCA Platform IPMI System Eventlog Error Event
MSMCAInfo_RawCMCEvent	This contains a CMC event
MSMCAInfo_RawCorrectedPlatformEvent	This contains a Corrected Platform event
MSMCAInfo_RawMCAData	This contains the raw MCA logs
MSMCAInfo_RawMCAEvent	This contains a MCA event
MSMouse_ClassInformation	Mouse class driver information
MSMouse_PortInformation	Mouse port driver information
MSNdis_CurrentLookahead	NDIS Current Lookahead
MSNdis_CurrentPacketFilter	NDIS Current Packet Filter
MSNdis_DeviceWakeOnMagicPacketOnly	This control decides whether the network device should wake up the system only on receiving a Magic packet
MSNdis_DriverVersion	NDIS Driver Version
MSNdis_EnumerateAdapter	NDIS Enumerate Adapter
MSNdis_HardwareStatus	NDIS Hardware Status
MSNdis_LinkSpeed	NDIS Link Speed
MSNdis_MacOptions	NDIS MAC Options
MSNdis_MaximumFrameSize	NDIS Maximum Frame Size
MSNdis_MaximumLookahead	NDIS Maximum Lookahead Supported
MSNdis_MaximumSendPackets	NDIS Maximum Send Packets
MSNdis_MaximumTotalSize	NDIS Maximum Packet Total Size
MSNdis_MediaConnectStatus	NDIS Media Connect Status
MSNdis_MediaInUse	NDIS Media Types In Use
MSNdis_MediaSupported	NDIS Media Types Supported
MSNdis_NdisEnumerateVc	NDIS Enumerate VC
MSNdis_NotifyAdapterArrival	NDIS Notify Adapter Arrival
MSNdis_NotifyAdapterRemoval	NDIS Notify Adapter Removal
MSNdis_NotifyVcArrival	NDIS Notify VC Arrival
MSNdis_NotifyVcRemoval	NDIS Notify VC Removal
MSNdis_PhysicalMediumType	NDIS Physical Medium Type
MSNdis_ReceiveBlockSize	NDIS Receive Block Size
MSNdis_ReceiveBufferSpace	NDIS Receive Buffer Space
MSNdis_ReceiveError	NDIS Receive Errors
MSNdis_ReceiveNoBuffer	NDIS Receive No Buffer
MSNdis_ReceivesOk	NDIS Receives OK
MSNdis_StatusDevicePowerOff	NDIS Device Power Off Notification
MSNdis_StatusDevicePowerOn	NDIS Device Power On Notification
MSNdis_StatusLinkSpeedChange	NDIS Status Link Speed Change
MSNdis_StatusMediaConnect	NDIS Status Media Connect
MSNdis_StatusMediaDisconnect	NDIS Status Media Disconnect
MSNdis_StatusMediaSpecificIndication	NDIS Status Media Specific Indication
MSNdis_StatusProtocolBind	NDIS Protocol Bind Notification
MSNdis_StatusProtocolUnbind	NDIS Protocol Unbind Notification
MSNdis_StatusResetEnd	NDIS Status Reset End
MSNdis_StatusResetStart	NDIS Status Reset Start
MSNdis_TransmitBlockSize	NDIS Transmit Block Size
MSNdis_TransmitBufferSpace	NDIS Transmit Buffer Space
MSNdis_TransmitsError	NDIS Transmit Errors
MSNdis_TransmitsOk	NDIS Transmits OK
MSNdis_VendorDescription	NDIS Vendor Description
MSNdis_VendorDriverVersion	NDIS Vendor's Driver Version
MSNdis_VendorID	NDIS Vendor ID
MSNdis_VlanIdentifier	NDIS VLAN Identifier

Table 5.17 *The WDM Providers Classes (continued)*

Name	Description
MSPower_DeviceEnable	The control sets whether the device should dynamically power on and off while the system is working.
MSPower_DeviceWakeEnable	This control indicates whether the device should be configured to wake a sleeping system.
MSRedbook_DriverInformation	Digital Audio Filter Driver Information (redbook)
MSRedbook_Performance	Digital Audio Filter Driver Performance Data (redbook)
MSSerial_CommInfo	Serial Communications Information
MSSerial_CommProperties	Communication properties for serial port
MSSerial_HardwareConfiguration	Hardware configuration for serial port
MSSerial_PerformanceInformation	Performance information for serial port
MSSerial_PortName	Serial Port Name
MSSmBios_RawSMBiosTables	Raw SMBIOS Tables
MSSmBios_SMBiosEventlog	Raw SMBIOS Eventlog
MSSmBios_Sysid1394List	List of 1394 SYSIDS
MSSmBios_SysidUUIDList	List of UUID SYSIDS
MSTapeDriveParam	Tape Drive Parameters
MSTapeDriveProblemEvent	Tape Drive Problem Warning
MSTapeMediaCapacity	Tape Media Capacity
MSTapeProblemDeviceError	Device Errors
MSTapeProblemIoError	IO Read Write Errors
MSTapeSymbolicName	Tape Symbolic Name
ProcessorCStateEvent	Processor CStates Event
ProcessorMethods	Methods to alter Processor Performance States
ProcessorPerformance	Processor Information
ProcessorPerfStateEvent	Processor Performance State Change Event
ProcessorStatus	Processor Performance Information
ProcessorTransitionEvent	Processor Transition Event

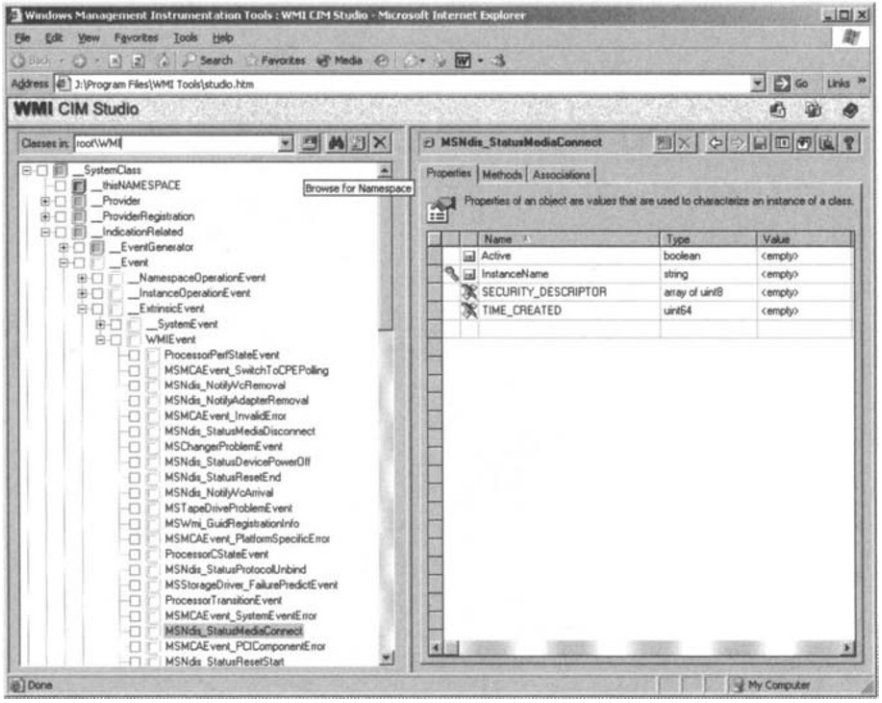

Figure 5.15 *The WMIEvent extrinsic event class.*

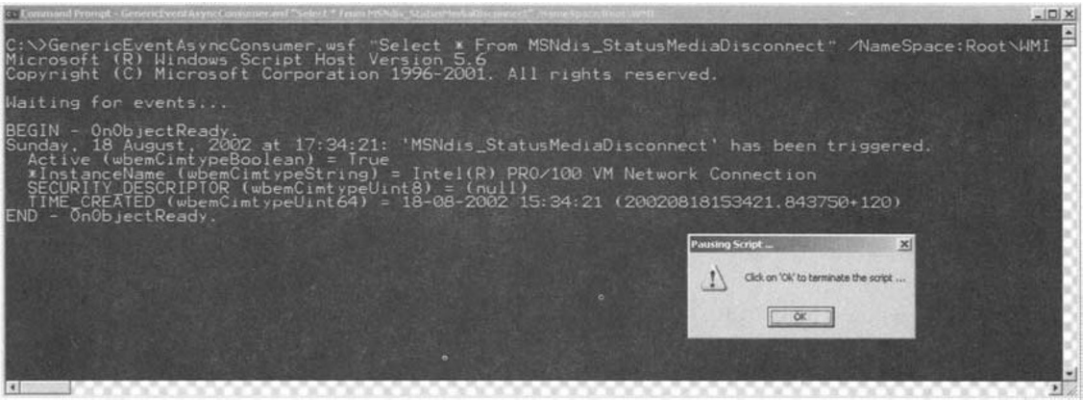

Figure 5.16 *Detecting network cable disconnections with the MSNdis_StatusMediaDisconnect extrinsic event class.*

Actually, the *MSDiskDriver_Geometry* class exposes a subset of the information retrieved by the *Win32_DiskDrive* class.

As a second example, we can also detect network cable disconnections/reconnections. To do so, we must work with the *MSNdis_StatusMediaDisconnect* class, which is a subclass of the *WMIEvent* extrinsic event class. If we reuse Sample 6.17 ("A generic script for asynchronous event notification") in the appendix, when a network cable disconnection occurs, the script returns an event notification, as shown in Figure 5.16.

By requesting instances of the *MSNdis_MediaConnectStatus* class, it is possible to retrieve the state of all network connections available in the system.

```
C:\>GetCollectionOfInstances.wsf MSNdis_MediaConnectStatus /Namespace:Root\WMI
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.


Active: ................................ TRUE
*InstanceName: ......................... WAN Miniport (Network Monitor)
NdisMediaConnectStatus: ................ 0

Active: ................................ TRUE
*InstanceName: ......................... WAN Miniport (IP)
NdisMediaConnectStatus: ................ 0

Active: ................................ TRUE
*InstanceName: ......................... Intel(R) PRO/100 VM Network Connection
NdisMediaConnectStatus: ................ 0
```

When the *NdisMediaConnectStatus* property is set to 0, the network is connected. When the property is set to 1, the network is disconnected.

5.3 WMI and some (server) products

5.3.1 Internet Information Server provider

The *Internet Information Server* (IIS) provider is available with IIS 6.0, which comes with Windows Server 2003. It is made up of one single provider implemented as a method, instance, and class provider. It is available from the **Root\MicrosoftIISv2** namespace (see Table 5.18).

This provider supports a collection of WMI classes whose purpose is to expose the information contained in the IIS metabase. The IIS metabase is nothing more than a database containing all IIS configuration settings. This information is stored in a collection of objects and properties defining the various settings supported by IIS. Of course, if we have objects and properties in the metabase, it means we have a schema defining those objects and properties. Therefore, as in any schema implementation, we also have objects with properties (part of the schema) defining objects and properties that can be stored in the metabase. These objects are called the Schema Management Objects. We will not go into the IIS metabase schema discovery, since this would detract from the WMI focus, but it is important to know that WMI classes reflect the IIS metabase schema definitions, whose object instances are represented by WMI instances. This is the reason why the *IIS* provider is implemented as a class provider (to retrieve the IIS metabase schema definitions as WMI classes) and an instance provider (to retrieve the IIS metabase data stored in the IIS metabase objects).

Understanding how WMI maps the IIS metabase is not an easy thing. So, let's take a concrete example. For instance, the IIS metabase schema contains definitions for the **IISWebServer** metabase class and its related

Table 5.18 *The Internet Information Server Provider*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
IIS Provider																
IIS_PROVIDER	Root/MicrosoftIISv2	X	X	X				X	X	X	X	X				

properties, such as the **ServerBindings** property. Therefore, the CIM repository contains some equivalent WMI classes and properties to expose the same information through WMI. In this specific example (see Figure 5.17), we will have an *IISWebServer* WMI class made from the *CIM_ManagedSystemElement* superclass (left-center position). However, the CIM repository object model is slightly different from the IIS metabase object model. For example, the *IISWebServer* WMI class doesn't have a property called *ServerBindings*. Instead, the *IISWebServer* WMI class has an association with the *IISWebServerSetting* WMI class made from the *CIM_Setting* superclass (right upper-corner position), which exposes a WMI class property called *ServerBindings*. The association between the *IISWebServer* and *IISWebServerSetting* WMI classes is made with the *IIsWebServer_IIsWebServerSetting* association class made from the *CIM_ElementSetting* class (center-top position).

The *ServerBindings* property of the *IISWebServerSetting* WMI class contains a WMI instance of an object made from the *ServerBinding* class

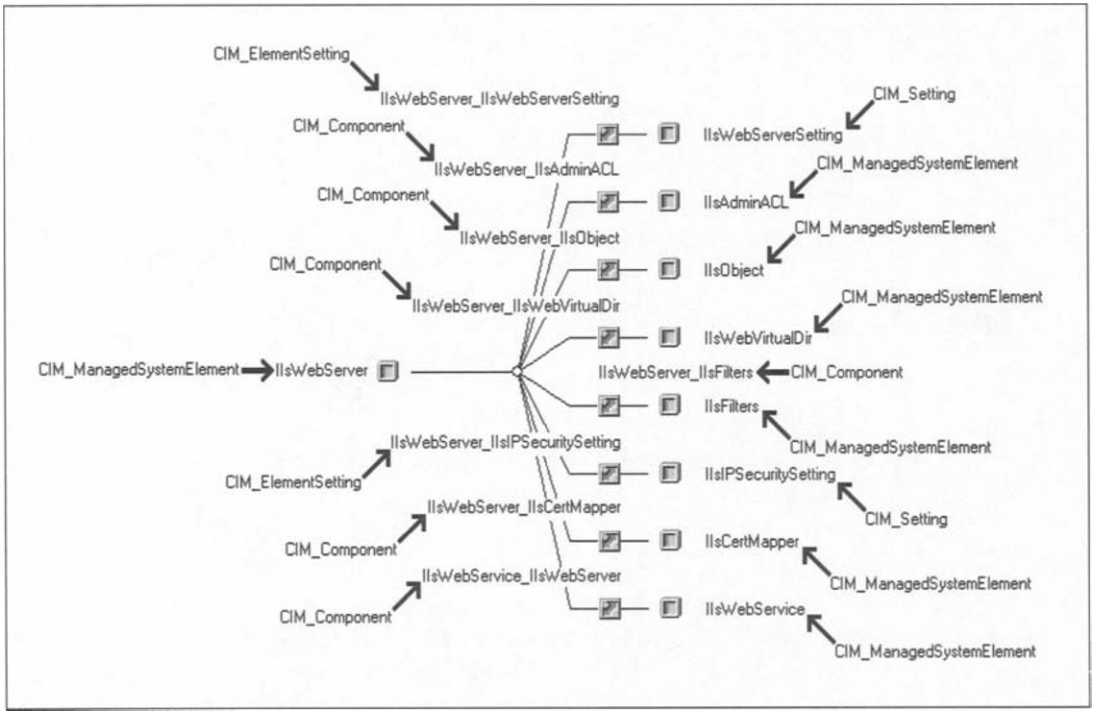

Figure 5.17 *The IISWebServer class associations with their respective superclasses.*

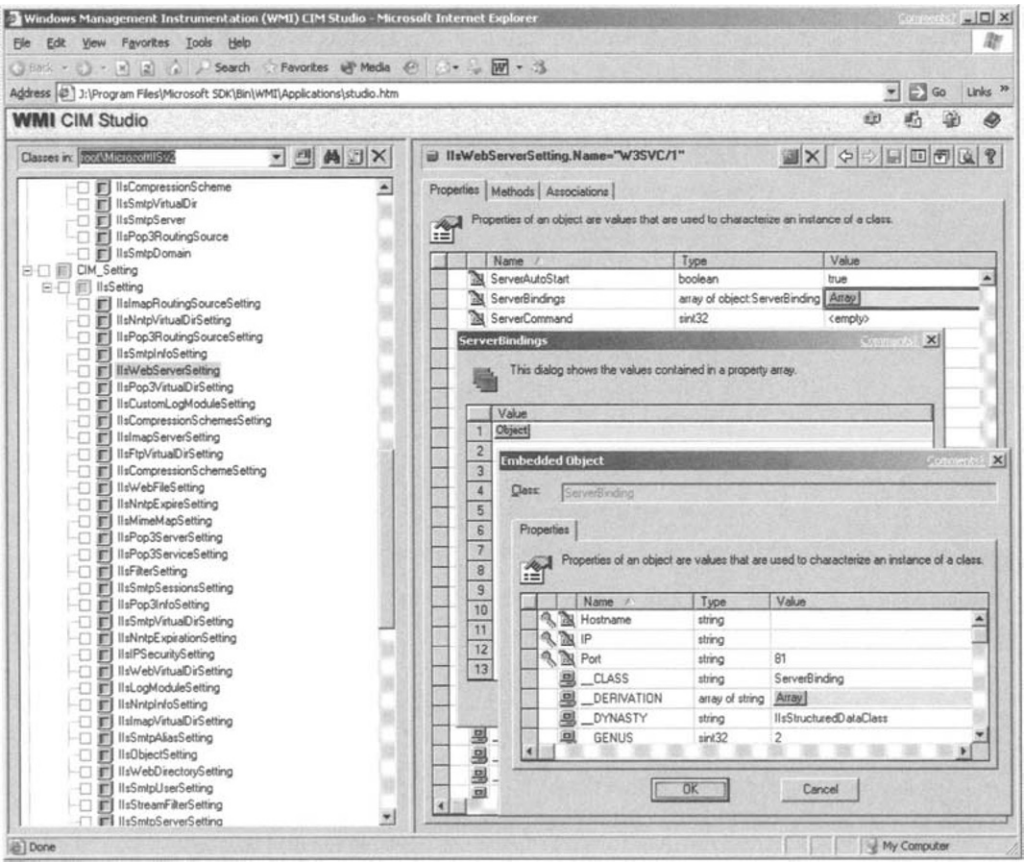

Figure 5.18 *The ServerBindings property and ServerBinding instance.*

(which is made from the *IisStructuredDataClass* superclass), as shown with **WMI CIM Studio** in Figure 5.18.

As we can see in Figure 5.17, the *IISWebServer* WMI class has some other associations. All associations are made from association classes derived from the *CIM_ElementSetting* or *CIM_Component* superclasses. On the other hand, all associated classes are made from the *CIM_Setting* or *CIM_ManagedSystemElement* superclasses. Next, when a WMI class exposing IIS settings requires a structured representation, the property that should expose the structured information actually exposes an instance made from a class derived from the *IIsStructuredDataClass* superclass (i.e., *ServerBinding* class). This indicates that all IIS WMI classes are always made from five superclasses (see Figure 5.17):

- **The *CIM_ManagedSystemElement* class:** The *CIM_ManagedSystemElement* class contains subclasses that correspond to node definitions of the metabase Schema. For example, an *IIsWebServer* WMI instance corresponds to the **IIsWebServer** node of the IIS metabase, which represents an instance of an IIS Web server. As another example, the *IIsWebVirtualDir* instance corresponds to the **IIsWebVirtualDir** node of the IIS metabase, which represents an instance of a Web virtual directory. The *IIsWebServer* WMI instance and the *IIsWebVirtualDir* instance expose read-only properties and methods, whereas the associated instances made from the *CIM_Setting* subclasses contain the writeable properties for the nodes. For example, the *IIsWebServer* class derived from the *CIM_ManagedSystemElement* class exposes read-only properties for an IIS Web server as well as methods that can modify some properties of the *IIsWebServerSetting* class instance and manage the IIS Web Server (i.e., start or stop the Web server).

  On the other hand, the *IIsWebServerSetting* class properties (derived from the *CIM_Setting* class and associated with the *IISWebServer* class) can be updated to modify the IIS Web server settings (see Figure 5.19).

- **The *CIM_Setting* class:** The classes derived from the *CIM_Setting* superclass expose properties corresponding to the metabase node properties that can be set at those nodes. The associated classes derived from the *CIM_ManagedSystemElement* class expose methods that manipulate the node properties. As previously mentioned, the *IIsWebServerSetting* class (derived from the *CIM_Setting* superclass) and the *IIsWebServer* class (derived from the CIM_*ManagedSystemElement* class) refer to Web sites on your Web server, where the *IIsWebServer* class contains the read-only properties of a Web site and

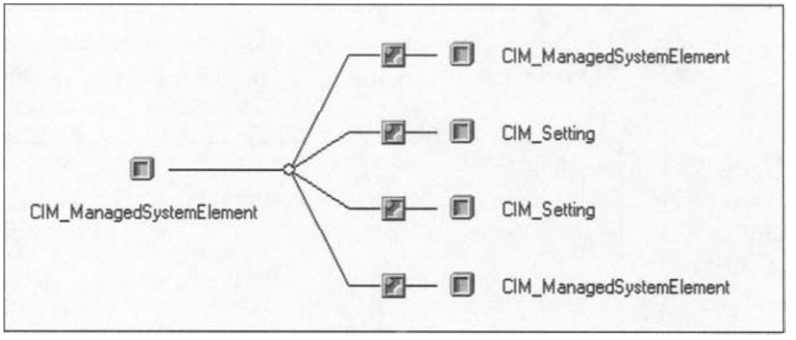

Figure 5.19 *The associations of the CIM_ManagedSystemElement superclass.*

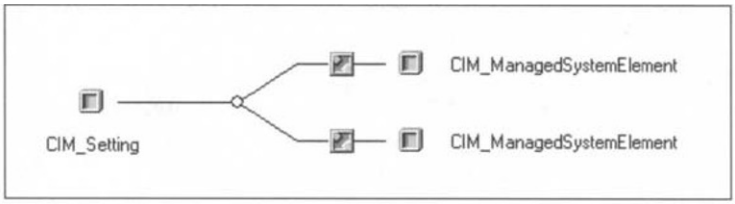

Figure 5.20 *The associations of the CIM_Setting superclass.*

the *IIsWebServerSetting* class contains the writeable properties of a Web site (see Figure 5.20).

- **The *IIsStructuredDataClass* class:** The classes derived from the *IIsStructuredDataClass* class contain properties whose data requires a structured representation. For example, the *ServerBindings* property in the metabase is a string whose format is "IP:Port:Hostname." In the WMI representation, a *ServerBinding* class is available with three properties corresponding to the IP address, the port number, and the host name. In Figure 5.18, a class such as the *IIsWebServerSetting* class (made from the *CIM_Setting* class) contains a property called *ServerBindings,* which is an array of *ServerBinding* class instances.
- **The *CIM_Component* class:** The classes created from this association superclass map each class made from the *CIM_ManagedSystemElement* superclass to another class made from the *CIM_ManagedSystemElement* superclass. As with any association classes, the properties of these classes are references to the two associated classes (see Figure 5.21).
- **The *CIM_ElementSetting* class:** The classes created from this association class associate each class made from the *CIM_ManagedSystemElement* superclass to its matching class made from the *CIM_Setting* superclass. The properties of these classes are references to the two associated classes (see Figure 5.22).

To read information from the metabase with scripts, it is possible to use ADSI in the IIS: namespace (all IIS versions) or WMI (IIS 6.0 only). A

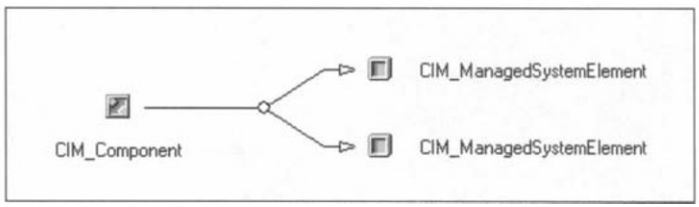

Figure 5.21 *The CIM_Component class with its references.*

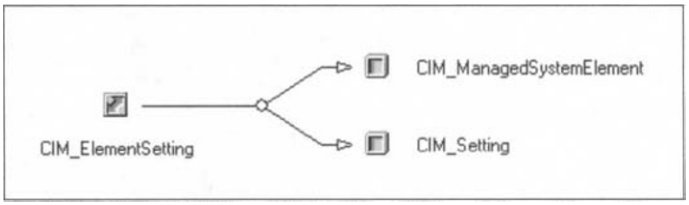

Figure 5.22 *The CIM_ElementSetting class with its references.*

script reading the **IISWebServer ServerBindings** property with ADSI will look as follows:

```
1:   Set objIISWebServer = GetObject ("IIS://LocalHost/W3SVC/1")
2:
3:   arrayServerBindings = objIISWebServer.Get ("ServerBindings")
4:
5:   For Each varServerBinding In arrayServerBindings
6:       WScript.Echo "ServerBindings: '" & varServerBinding & "'"
7:   Next
```

At line 1, the "W3SVC/1" represents the IIS Web server instance to be accessed via ADSI. Next, the script requests the **ServerBindings** property from the object representing the Web service instance (line 3). From line 5 through 7, the "For Each" loop enumerates all values found in the **ServerBindings** property.

The **ADSUTIL.VBS** script from an IIS installation (check %SystemDrive%\Inetpub\AdminScripts folder) uses ADSI to access the IIS metabase. For example, to read the **IISWebServer ServerBindings** property with **ADSUTIL.VBS**, we should use the following command line:

```
C:\>adsutil.vbs GET W3SVC/1/ServerBindings
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

ServerBindings                    : (LIST)  (1 Items)
  ":81:"
```

In addition to being able to retrieve all properties stored in the IIS metabase, **ADSUTIL.VBS** also allows most configuration settings and operations handled by the "Internet Information Services" MMC from the command line.

With WMI, to retrieve the **ServerBindings** of the **IISWebServer** "W3SVC/1" node from the metabase, we should retrieve the *IISWebServerSettings* WMI instance (which is actually associated with the *IISWebServer* WMI class and contains the Web server configuration settings). This is illustrated in Sample 5.15. However, for information completeness, the

code enumerates all properties of the *IISWebServerSettings* WMI instance, which explains why it is slightly longer than the ADSI example.

Sample 5.15 *Viewing the IISWebServer ServerBindings property with WMI*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
..:
17:    </runtime>
18:
19:    <script language="VBScript" src="..\Functions\DisplayFormattedPropertiesFunction.vbs" />
20:    <script language="VBScript" src="..\Functions\DisplayFormattedPropertyFunction.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\MicrosoftIISv2"
32:    Const cWMIClass = "IISWebServer"
..:
59:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
60:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
61:
62:    Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
63:                                                     strUserID, strPassword)
..:
66:    Set objWMIWebServerInstances = objWMIServices.InstancesOf (cWMIClass)
..:
69:    For Each objWMIWebServerInstance In objWMIWebServerInstances
70:        DisplayFormattedProperties objWMIWebServerInstance, 0
71:
72:        Set objWMIAssociatedInstances = objWMIServices.ExecQuery _
73:                                        ("Associators Of {IISWebServer='" & _
74:                                        objWMIWebServerInstance.Name & "'}")
75:        For Each objWMIAssociatedInstance In objWMIAssociatedInstances
76:            DisplayFormattedProperties objWMIAssociatedInstance, 2
77:        Next
..:
79:    Next
..:
85:    ]]>
86:    </script>
87:  </job>
88:</package>
```

Sample 5.15 does not use any new scripting techniques. However, it exploits the *IISWebServer* class associations in place to retrieve all required information. First, the script retrieves all instances of the *IISWebServer* WMI class (line 66) to enumerate all instances retrieved (lines 69 through

77). For each instance of the *IISWebServer* class, the script requests all existing associations (lines 72 through 74), as shown in Figure 5.17. This is done with the help of the WQL data query:

```
Associators Of {IISWebServer='W3SVC/1'}
```

To restrict the list of associations to the *IISWebServerSetting* class, the following WQL data query can be used:

```
"Associators Of {IISWebServer='W3SVC/1'} Where AssocClass=IIsWebServer_IIsWebServerSetting"
```

Next all instances are enumerated to display all their properties (lines 75 through 77). The display of the properties is performed in the DisplayFormattedProperties() function. This function acts recursively if a property, such as the *ServerBindings* property of the *IISWebServerSetting* class, exposes an object instance. Therefore, the output will be as follows for the first IIS Web server instance:

```
  1:  C:\>ViewIISSettings.wsf
  2:  Microsoft (R) Windows Script Host Version 5.6
  3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
  4:
  5:
  6:  - IIsWebServer --------------------------------------------------------------------
  7:  AppIsolated: ............................. 2
  8:  AppPackageID: ............................
  9:  AppPackageName: ..........................
 10:  *Name: ................................... W3SVC/1
 11:  ServerState: ............................. 2
 12:
 13:    - IIsWebServerSetting ---------------------------------------------------------
 14:    AccessExecute: ......................... FALSE
 15:    AccessFlags: ........................... 1
...:
146:    EnableDocFooter: ....................... FALSE
147:    EnableReverseDns: ...................... FALSE
148:    FrontPageWeb: .......................... FALSE
149:
150:      - HttpCustomHeader ------------------------------------------------------------
151:      *Keyname: ............................
152:
153:
154:      - HttpError -------------------------------------------------------------------
155:      *HandlerLocation: ..................... J:\WINDOWS\help\iisHelp\common\400.htm
156:      *HandlerType: ......................... FILE
157:      *HttpErrorCode: ....................... 400
158:      *HttpErrorSubcode: .................... *
...:
334:      - HttpError -------------------------------------------------------------------
335:      *HandlerLocation: ..................... J:\WINDOWS\help\iisHelp\common\500-15.htm
336:      *HandlerType: ......................... FILE
337:      *HttpErrorCode: ....................... 500
338:      *HttpErrorSubcode: .................... 15
339:
340:    HttpExpires: ........................... D, 0x15180
341:    HttpPics: ............................
```

```
...:
379:     MaxEndpointConnections: ..... 255
380:
381:       - MimeMap ------------------------------------------------------------------
382:       *Extension: ..........................
383:
384:     *Name: ................................ W3SVC/1
385:     NetLogonWorkstation: ................... 0
386:     NotDeletable: .......................... FALSE
...:
396:     RevocationFreshnessTime: ............... 128
397:     RevocationURLRetrievalTimeout: ......... 0
398:
399:       - ScriptMap ----------------------------------------------------------------
400:       *Extensions: ......................... .asp
401:       *Flags: .............................. 1
...:
556:       - SecureBinding ------------------------------------------------------------
557:       *IP: ..............................
558:       *Port: ............................... 443:
559:
560:     ServerAutoStart: ....................... TRUE
561:
562:       - ServerBinding ------------------------------------------------------------
563:       *Hostname: ...........................
564:       *IP: ..............................
565:       *Port: ............................... 80
566:
567:     ServerComment: ......................... Default Web Site
568:     ServerListenBacklog: ................... 40
...:
577:     UseHostName: ........................... FALSE
578:     Win32Error: ............................ 0
579:
580:   - IIsWebServer -------------------------------------------------------------
581:   AppIsolated: ............................. 2
582:   AppPackageID: ............................
583:   AppPackageName: .........................
584:   *Name: ................................... W3SVC/2
585:   ServerState: ............................. 2
...:
...:
...:
```

We recognize from line 562 through 565 the properties of the *ServerBinding* class instance. Of course, Sample 5.15 retrieves all available properties. To just retrieve the **IISWebServer ServerBindings** property, such as we did with ADSI, the code to use will look as follows:

```
 1:  Set objWMIServices = GetObject ("winmgmts:root\MicrosoftIISv2")
 2:
 3:  Set objWMIWebServerInstance = objWMIServices.Get ("IISWebServerSetting='W3SVC/1'")
 4:  arrayWMIServerBindings = objWMIWebServerInstance.ServerBindings
 5:
 6:  For Each objWMIServerBindingInstance In arrayWMIServerBindings
 7:      WScript.Echo objWMIServerBindingInstance.Port
 8:      WScript.Echo objWMIServerBindingInstance.Hostname
 9:      WScript.Echo objWMIServerBindingInstance.IP
10:  Next
```

Even if it is possible to perform many IIS metabase configurations through ADSI or WMI, the ADSI metabase access method has some limitations compared with the WMI access method. For example, the WMI access method implicitly offers all advantages of the WMI architecture, such as the WQL queries, the standard WMI COM API object model and instrumentation (i.e., Event notifications), and the CIM repository object model capabilities to associate entities. ADSI doesn't have these features. On the other hand, it is important to note that only ADSI allows you to extend the IIS metabase. However, once the extensions are created, the IIS WMI provider can return existing schema extensions. This can be done because the *IIS* provider is also implemented as a class provider.

Even if the IIS metabase object model is slightly different from the WMI class object model, it is clear that a good knowledge of the IIS metabase is an indispensable foundation in understanding the meaning of each WMI class representing manageable IIS entities. The **Root\MicrosoftIISv2** namespace contains more than 300 WMI classes related to the IIS management, which makes it impossible to review them one by one in this section. Moreover, all these classes reflect the IIS metabase schema. However, with the Internet Information Server 6.0 in Windows Server 2003, Microsoft has created a very nice *IIS* Provider Tutorial. It is available from the Internet Information Server MMC by selecting "Help" and "Help Topics." This tutorial is located in the "Programmatic Administration Guide" and is entitled the "IIS WMI Provider Tutorial." It is also possible to access the tutorial directly from the IISMMC.CHM help file located in the %SystemRoot%\System32\Help directory. For anyone interested in delving into the IIS management from WMI, this tutorial is worth reading! Of course, you can also continue to use the **LoadCIMinXL.Wsf** (see Sample 4.32 in the appendix) to retrieve detailed information about the IIS WMI classes.

5.3.2 Exchange 2000

With the release of Exchange 2000, Microsoft included three WMI providers: *ExchangeRoutingTableProvider*, *ExchangeQueueProvider*, and *ExchangeClusterProvider*, which provide an easy way for any application to access Exchange 2000 management information. Each of the three providers relates to a specific component set of Exchange 2000:

- The *ExchangeRoutingTableProvider* runs on top of the routing API.
- The *ExchangeQueueProvider* runs on top of the queue API.
- The *ExchangeClusterProvider* runs on top of the cluster API.

Table 5.19 *The Exchange 2000 WMI Providers*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Exchange 2000 RTM	Exchange 2000 SP1	Exchange 2000 SP2
Exchange 2000 Providers														
Exchange Queue provider	Root/CIMv2/Applications/Exchange		X							X		X	X	
Exchange Cluster provider	Root/CIMv2/Applications/Exchange		X							X		X	X	
Exchange Routing Table provider	Root/CIMv2/Applications/Exchange		X						X	X		X	X	
Exchange Message Tracking Log provider	root/MicrosoftExchangeV2		X					X	X	X	X			X
Exchange DS Access provider	root/MicrosoftExchangeV2		X					X		X				X

Each of these providers delivers information with a set of classes available in the **Root\CIMv2\Applications\Exchange** namespace to ease the notification and diagnostics of some problems that may occur in Exchange 2000. With the release of Service Pack 2, Exchange 2000 delivers two additional WMI providers:

- The *Exchange Message Tracking* provider runs on top of the message tracking API.
- The *Exchange DS Access* provider runs on top of the DSAccess API.

These two providers expose a set of classes available from the **Root\MicrosoftExchangeV2** WMI namespace (see Table 5.19).

The release of Exchange 2000 Service Pack 3 does not add anything new regarding WMI.

Note that none of these providers is implemented as event providers. This will imply the use of the **WITHIN** statement when executing WQL event queries. Moreover, as we will see in the following text, none of the classes supported by these providers is associated with any other classes. They are implemented as standalone classes providing information about some specific Exchange components.

5.3.2.1 The Routing Table provider

The *Exchange Routing Table* WMI provider works on top of the Exchange Transport Core. You may think that the purpose of this provider is to access available existing routes from the routing table of Exchange 2000, but you'd be wrong. Rather, its purpose is as follows:

- To publish the status of the local Exchange 2000 server in the routing table, based on the monitoring conditions configured with the Exchange System Manager (ESM).

- To retrieve the status of other Exchange 2000 servers in the organization from the routing table, based on the monitoring conditions configured with the ESM.
- To publish the status of the Exchange 2000 local connectors in the routing table.
- To retrieve the status of other Exchange 2000 connectors present in the organization from the routing table.

The routing table is used as a transport to publish the local status of the server and the local connector states in the organization. To perform this task, the *Exchange Routing Table* provider implements a dedicated way of access to and from the routing table only for the System Attendant. Every Exchange 2000 server in the organization is publishing its state in the routing table in this way. This means that it is possible for one server to get the status of all the servers and connectors in the enterprise as they are seen from that server. To distinguish the information retrieved between the server and the connectors, the *Exchange Routing Table* provider implements two WMI classes: *ExchangeServerState* and *ExchangeConnectorState*. Each of these classes has a specific set of properties.

The states retrieved from the *ExchangeServerState* class are based on the monitoring conditions set with the ESM. Different information states are available from this class for the critical components of an Exchange 2000 server installation. The ESM allows an Administrator to configure a set of conditions determining a state for the monitored com-

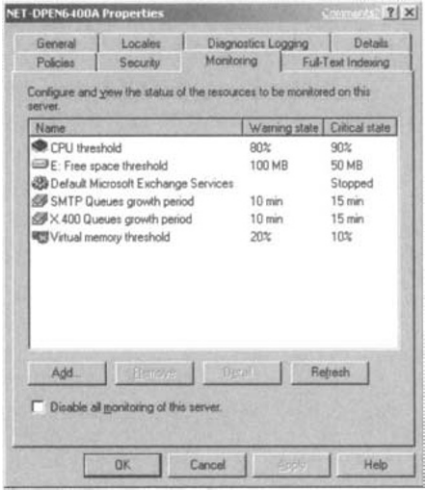

Figure 5.23 *The Exchange System Manager monitoring settings.*

ponent (see Figure 5.23). With the ESM, it is possible to define a monitoring condition with a corresponding state for:

- The queues
- The disk space available on any disk in the system
- The memory usage
- The CPU usage for a "warning" and/or a "critical" threshold
- Any Windows 2000 services relevant to Exchange 2000

The *ExchangeConnectorState* class is based on the same principle as the *ExchangeServerState* class. The class provides a monitoring capability of the connectors configured in an Exchange organization. The state is published in the routing table. When this class is interrogated from the ESM or by a script, the class offers a list of the connectors available from the routing table with their corresponding status. Using this class, it is possible to know the current state of all the connectors in the Exchange 2000 organization as seen from that server (see Figure 5.24).

The set of properties exposed by the *ExchangeServerState* class can be retrieved with the **LoadCIMinXL.Wsf** script (see Sample 4.32 in the appendix). If we reuse the **GetCollectionOfInstances.wsf** script (see Sample

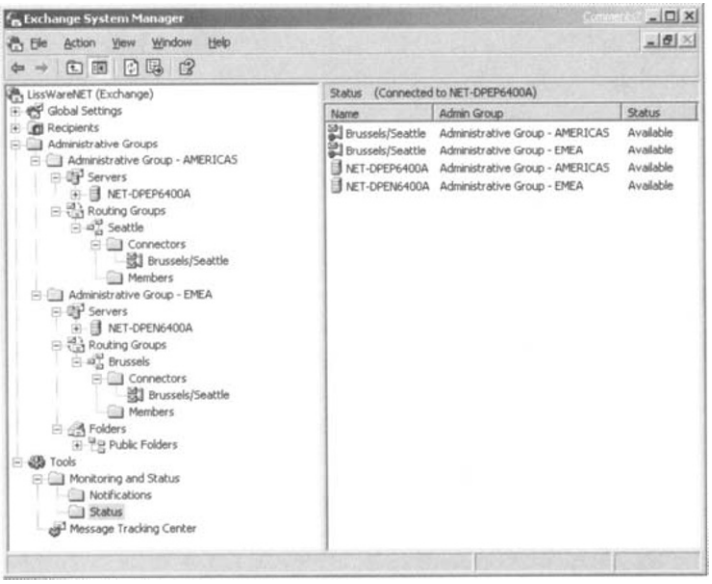

Figure 5.24 *The Exchange System Manager showing the Exchange servers and connectors state.*

1.5, "Listing all instances of a class with their properties formatted"), we obtain the following output for the *ExchangeServerState* class:

```
C:\>GetCollectionOfInstances.wsf ExchangeServerState
                                 /NameSpace:Root\CIMV2\Applications\Exchange
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.


ClusterState: ........................... 1
ClusterStateString: ..................... OK
CPUState: ............................... 1
CPUStateString: ......................... OK
DisksState: ............................. 1
DisksStateString: ....................... OK
*DN: .................................... CN=NET-DPEN6400A,CN=Servers,
                                          CN=Administrative Group - EMEA,
                                          CN=Administrative Groups,CN=LisswareNET,
                                          CN=Microsoft Exchange,CN=Services,
                                          CN=Configuration,DC=LisswareNET
GroupDN: ................................ CN=Seattle,CN=Routing Groups,
                                          CN=Administrative Group - EMEA,
                                          CN=Administrative Groups,CN=LisswareNET,
                                          CN=Microsoft Exchange,CN=Services,
                                          CN=Configuration,DC=LisswareNET
GroupGUID: .............................. {088D875C-335A-4429-807F-B8B257CE15DE}
GUID: ................................... {B5ED627E-E59A-4A43-8B6C-F71E5F2CEC6E}
MemoryState: ............................ 1
MemoryStateString: ...................... OK
Name: ................................... NET-DPEN6400A
QueuesState: ............................ 1
QueuesStateString: ...................... OK
ServerMaintenance: ...................... FALSE
ServerState: ............................ 1
ServerStateString: ...................... OK
ServicesState: .......................... 1
ServicesStateString: .................... OK
Unreachable: ............................ FALSE
Version: ................................ 6132
```

Table 5.20 lists the *ExchangeServerState* class properties.

5.3.2.2 The Queue provider

The WMI *Queue* provider is based on the Queue API. The scope of this provider is local to the Exchange 2000 server. The provider implements two WMI classes:

- The *ExchangeLink* class to retrieve information about the Exchange links directly from the Queue API
- The *ExchangeQueue* class to retrieve information about the Exchange queues directly from the Queue API

Table 5.20 *The ExchangeServerState Class Properties*

Name	Description
ClusterState Property	When the ExchangeServerState instance represents a clustered Exchange server, the ClusterState property specifies the state of the clustered resources on that server.
ClusterStateString Property	When the ExchangeServerState instance represents a clustered Exchange server, the ClusterStateString property specifies the state of the cluster resources on that server.
CPUState Property	The CPUState property specifies the current state of the CPU on the Exchange server. This is the same state information shown on the Monitoring and Status Properties page of the Exchange System Manager.
CPUStateString Property	The CPUStateString property specifies the current state of the CPU on the Exchange server.
DisksState Property	The DisksState property specifies the current state of the disk storage on the computer running Exchange 2000 Server.
DisksStateString Property	The DisksStateString property specifies the current state of the disk storage on the computer running Exchange 2000 Server.
DN Property	The DN property specifies the Microsoft Active Directory distinguished name (DN) of the Exchange server object.
GroupDN Property	The GroupDN property specifies the DN of the Exchange 2000 Server routing group in Active Directory.
GroupGUID Property	The GroupGUID property specifies the globally unique identifier (GUID) of the Exchange 2000 Server routing group in Active Directory.
GUID Property	The GUID property specifies the GUID of the Exchange 2000 Server server object in Active Directory.
MemoryState Property	The MemoryState property specifies the current state of the memory on the computer running Exchange 2000 Server.
MemoryStateString Property	The MemoryStateString property specifies the current state of the memory on the computer running Exchange 2000 Server.
Name Property	The Name property specifies the name of the computer running Exchange 2000 Server.
QueuesState Property	The QueuesState property specifies the current state of the queues on the computer running Exchange 2000 Server.
QueuesStateString Property	The QueuesStateString property specifies the current state of the queues on the computer running Exchange 2000 Server.
ServerMaintenance Property	The ServerMaintenance property, when TRUE, specifies that the notifications set up in the Exchange 2000 Server System Manager Monitoring and Status page have been disabled.
ServerState Property	The ServerState property specifies the current state of the computer running Exchange 2000 Server.
ServerStateString Property	The ServerStateString property specifies the current state of the computer running Exchange 2000 Server.
ServicesState Property	The ServicesState property specifies the current state of the monitoring services running on the Exchange 2000 Server computer.
ServicesStateString Property	The ServicesStateString property specifies the current state of the monitoring services running on the Exchange 2000 Server computer.
Unreachable Property	The Unreachable property, when TRUE, specifies that the Exchange 2000 Server computer is currently unreachable.
Version Property	The Version property indicates the version of the Exchange server.

For both classes, most of the properties are the ones exposed by the Queue API.

From a scripting and management perspective, for both classes, the most interesting data is the *IncreasingTime* property. The Queue API does not provide this property directly. This is a property calculated by the WMI *Queue* provider (see Figure 5.25).

The *IncreasingTime* value represents the length of time, while the number of messages in the Link/Queue has not decreased. The time is returned in milliseconds. To monitor this property, it is important to understand how the calculation is made. Two important factors influence how to monitor the value:

- The sampling frequency (Tx interval in Figure 5.25)
- The *IncreasingTime* value threshold to determine a critical situation

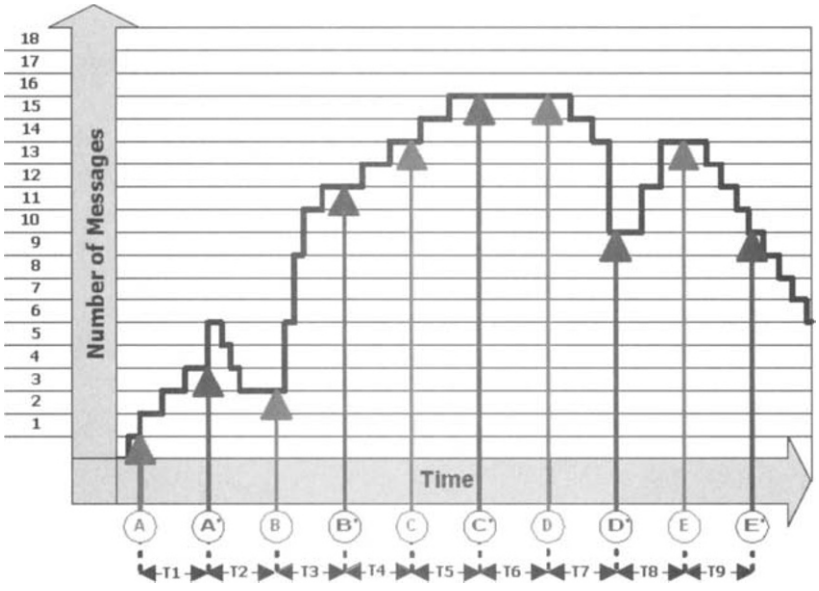

Figure 5.25 *The IncreasingTime property behavior from the ExchangeLink and ExchangeQueue classes.*

Each time a sample is taken, the *IncreasingTime* property is calculated and a new value is determined. The first polling interval (A in Figure 5.25) always returns an *IncreasingTime* value of zero, because it is the first sample read. But when the second polling interval occurs (A′ in Figure 5.25), the *IncreasingTime* value will be equal to the T1 interval, because the number of messages has not decreased between A and A′.

When the next polling interval occurs (B in Figure 5.25), the *IncreasingTime* value will be equal to zero, because the number of messages has decreased between A′ and B. It is important to note that the number of messages has decreased below the prior measured level. If the number of message levels does not decrease below the prior level, the *IncreasingTime* value is equal to T1 + T2. If the number of messages decreases to below the prior level, then the *IncreasingTime* value is set to zero.

For the same situation, if the polling interval is changed to measure at point A for the first sample and at point C for the second sample, the *IncreasingTime* value will not be equal to zero, because the number of messages has not been detected as decreasing below the number of messages measured at point A. The *IncreasingTime* value will be equal to T1 + T2 + T3 + T4.

Furthermore, if the polling is done between:

- A and B: *IncreasingTime* is equal to T1 + T2.

- B and C: *IncreasingTime* is equal to T1 + T2 + T3 + T4.
- C and D: *IncreasingTime* is equal to T1 + T2 + T3 + T4 + T5 + T6.
- D and E: *IncreasingTime* is equal to zero.

Now, if the polling is done in the following way:

- A and A′: *IncreasingTime* is equal to T1.
- A′ and B: *IncreasingTime* is equal to zero.
- B and B′: *IncreasingTime* is equal to T3.
- B′ and C: *IncreasingTime* is equal to T3 + T4.
- C and C′: *IncreasingTime* is equal to T3 + T4 + T5.
- C′ and D: *IncreasingTime* is equal to T3 + T4 + T5 + T6.
- D and D′: *IncreasingTime* is equal to zero.
- D′ and E: *IncreasingTime* is equal to T8.
- E and E′: *IncreasingTime* is equal to zero.

By changing the sampling frequency, we see that the *IncreasingTime* value can be very different. With the slowest sampling frequency, we miss the decreasing period between A and B. With the fastest polling frequency, this decreasing period is detected. Now, it is important to look at this behavior in a real-time scale.

If we suppose that the Tx interval is equal to one minute, the alert threshold for the *IncreasingTime* value is set on 30,000 ms (30 s), and the polling interval is set every Tx (every one minute), you will get a notification at point A′. This will be useless, because the queue has decreased between A′ and B. It is clear that you don't want to be notified for such a situation. In a production environment the number of messages can increase suddenly for one minute (or more), but the number of messages can decrease rapidly in the next few minutes. In such a case, you get an alert for a noncritical situation.

Now, if we suppose that the Tx interval is equal to one hour, the alert threshold for the *IncreasingTime* value is set on 14,400,000 ms (4 h), and the polling interval is set every Tx (every one hour), you will only get a notification at point D. This is a better situation, because you are notified for a nondecreasing situation after four hours. On the other hand, it is a little bit too long to be notified, because this situation may hide a real problem.

The key is to find a compromise between the polling intervals to detect quickly any nondecreasing situations without getting an alert for increasing

situations that are temporary. A best practice is to choose, for instance, to poll every 10 s and an *IncreasingTime* threshold value of 1,800,000 ms (30 min). In this situation, the flow represented in Figure 5.25 will never generate an alert. Only real nondecreasing situations longer than a half-hour will be detected with a precision of 10 s.

Again, you can use the **LoadCIMinXL.Wsf** script (see Sample 4.32 in the appendix) to discover the *ExchangeLink* and *ExchangeQueue* class properties. To get a notification if the *IncreasingTime* reaches a fixed value, the following WQL event query can be used:

```
C:\>GenericEventAsyncConsumer.wsf "Select * From __InstanceModificationEvent Within 5
                                   Where TargetInstance ISA 'ExchangeQueue' And
                                   TargetInstance.IncreasingTime > 10000"
                                   /Namespace:root\CIMV2\Applications\Exchange
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Waiting for events...

BEGIN - OnObjectReady.
Saturday, 22 June, 2002 at 09:33:17: '__InstanceModificationEvent' has been triggered.
  PreviousInstance (wbemCimtypeObject)
    CanEnumAll (wbemCimtypeBoolean) = True
    CanEnumFailed (wbemCimtypeBoolean) = True
    CanEnumFirstNMessages (wbemCimtypeBoolean) = True
    CanEnumFrozen (wbemCimtypeBoolean) = True
    CanEnumInvertSense (wbemCimtypeBoolean) = True
    CanEnumLargerThan (wbemCimtypeBoolean) = True
    CanEnumNLargestMessages (wbemCimtypeBoolean) = False
    CanEnumNOldestMessages (wbemCimtypeBoolean) = False
    CanEnumOlderThan (wbemCimtypeBoolean) = True
    CanEnumRecipient (wbemCimtypeBoolean) = True
    CanEnumSender (wbemCimtypeBoolean) = True
    GlobalStop (wbemCimtypeBoolean) = False
    IncreasingTime (wbemCimtypeUint32) = 5227
    *LinkName (wbemCimtypeString) = CurrentlyUnreachableLink
    MsgEnumFlagsSupported (wbemCimtypeUint32) = -1073741505
    NumberOfMessages (wbemCimtypeUint32) = 1
    *ProtocolName (wbemCimtypeString) = SMTP
    *QueueName (wbemCimtypeString) = net-dpep6400a.Emea.LissWare.NET
    SizeOfQueue (wbemCimtypeUint64) = 243
    Version (wbemCimtypeUint32) = 4
    VirtualMachine (wbemCimtypeString) = NET-DPEN6400A
    *VirtualServerName (wbemCimtypeString) = 1
  SECURITY_DESCRIPTOR (wbemCimtypeUint8) = (null)
  TargetInstance (wbemCimtypeObject)
    CanEnumAll (wbemCimtypeBoolean) = True
    CanEnumFailed (wbemCimtypeBoolean) = True
    CanEnumFirstNMessages (wbemCimtypeBoolean) = True
    CanEnumFrozen (wbemCimtypeBoolean) = True
    CanEnumInvertSense (wbemCimtypeBoolean) = True
    CanEnumLargerThan (wbemCimtypeBoolean) = True
    CanEnumNLargestMessages (wbemCimtypeBoolean) = False
    CanEnumNOldestMessages (wbemCimtypeBoolean) = False
    CanEnumOlderThan (wbemCimtypeBoolean) = True
```

```
43:       CanEnumRecipient (wbemCimtypeBoolean) = True
44:       CanEnumSender (wbemCimtypeBoolean) = True
45:       GlobalStop (wbemCimtypeBoolean) = False
46:       IncreasingTime (wbemCimtypeUint32) = 10415
47:       *LinkName (wbemCimtypeString) = CurrentlyUnreachableLink
48:       MsgEnumFlagsSupported (wbemCimtypeUint32) = -1073741505
49:       NumberOfMessages (wbemCimtypeUint32) = 1
50:       *ProtocolName (wbemCimtypeString) = SMTP
51:       *QueueName (wbemCimtypeString) = net-dpep6400a.Emea.LissWare.NET
52:       SizeOfQueue (wbemCimtypeUint64) = 243
53:       Version (wbemCimtypeUint32) = 4
54:       VirtualMachine (wbemCimtypeString) = NET-DPEN6400A
55:       *VirtualServerName (wbemCimtypeString) = 1
56:     TIME_CREATED (wbemCimtypeUint64) = (null)
57:   END - OnObjectReady.
```

As we can see, the *IncreasingTime* value is higher than 10 s between the **PreviousInstance** (line 22) and the **TargetInstance** (line 46). Tables 5.21 and 5.22 list the *ExchangeLink* and *ExchangeQueue* properties, respectively.

5.3.2.3 The Cluster provider

The *Cluster* provider implements the *ExchangeClusterResource* class directly, based on the cluster service. The State property contains the state of the Exchange 2000 Cluster group. This class is similar in function to the *ExchangeServerState*, but it works at the cluster level instead of the server level. Table 5.23 lists the *ExchangeClusterResource* properties.

5.3.2.4 The Message Tracking Logs provider

When "Message Tracking" is enabled on an Exchange 2000 server, all messages that go through this server are logged in a shared directory. The content of this directory can be examined with the Exchange System Manager, via the "Message Tracking Center," as shown in Figure 5.26.

With the tracking center, it is possible to track messages by sender, target recipients, server name, and message ID. It is also possible to select a window time (start date, end date) for the selection.

Before Exchange 2000 Service Pack 2, this was the only easy way to retrieve this information. Once Service Pack 2 is installed, it is possible to retrieve the same information with WMI.

The *Message Tracking Logs* WMI provider is called the *ExchangeMessageTrackingProvider* and gives access in read-only mode to the messages logged in the "Exchange Message Tracking" system. This provider supports one single class, called the *Exchange_MessageTrackingEntry* class (see Tables 5.24 and 5.25), which is a template representing a message tracking entry. With

Table 5.21 *The ExchangeLink Properties*

Name	Description
ActionFreeze Property	The ActionFreeze property, when TRUE, specifies that the link supports freezing messages in its queues. The ActionFreeze property corresponds to the sixth bit (0x00000020) of the SupportedLinkActions property.
ActionKick Property	The ActionKick property, when TRUE, specifies that the link can trigger its queues to retry transmitting waiting messages immediately, instead of waiting for the default protocol timeout before retrying the transmission. The ActionKick property corresponds to the first bit (0x00000001) of the SupportedLinkActions property.
ActionThaw Property	The ActionThaw property, when TRUE, specifies that the link supports thawing messages in its queues. Thawing a queue is also known as "unfreezing" that queue. The ActionThaw property corresponds to the seventh bit (0x00000040) of the SupportedLinkActions property.
ExtendedStateInfo Property	The ExtendedStateInfo property specifies the text description of the current link status.
GlobalStop Property	The GlobalStop property specifies whether the link is currently stopped.
IncreasingTime Property	The IncreasingTime property specifies the amount of time, in milliseconds, that the number of messages waiting to be transferred by the link has been increasing.
LinkDN Property	The LinkDN property specifies the Active Directory globally unique identifier (GUID) of the connector object that generated the link.
LinkName Property	The LinkName property specifies the name of the link.
NextScheduledConnection Property	The NextScheduledConnection property specifies the date and time when a connection will be attempted to transfer waiting messages.
NumberOfMessages Property	The NumberOfMessages property specifies the number of messages that are waiting for transmission across the link.
OldestMessage Property	The OldestMessage property specifies the date and time that the oldest message that is still waiting to be transmitted was received into the link.
ProtocolName Property	The ProtocolName property specifies the transmission protocol for the link.
SizeOfQueue Property	The SizeOfQueue property specifies the total size of the messages in the link, in bytes.
StateActive Property	The StateActive property, when TRUE, specifies that the link is active. The StateActive property corresponds to first bit (0x00000001) of the StateFlags property.
StateFlags Property	The StateFlags property specifies the state of the link. The individual bits of this property are available as the link State... and Type... properties of this class.
StateFrozen Property	The StateFrozen property indicates whether the link is currently frozen. The StateFrozen property corresponds to the sixth bit (0x00000020) of the StateFlags property.
StateReady Property	The StateReady property, when TRUE, specifies that the link is ready to accept new messages. The StateReady property corresponds to the second bit (0x00000002) of the StateFlags property.
StateRemote Property	The StateRemote property, when TRUE, specifies that the destination for messages in this link is on a remote server, instead of the messages being delivered to a local store. The StateRemote property corresponds to the fifth bit (0x00000010) of the StateFlags property.
StateRetry Property	The StateRetry property, when TRUE, specifies that the link is retrying a transmission that was unsuccessful. The StateRetry property corresponds to the third bit (0x00000004) of the StateFlags property.
StateScheduled Property	The StateScheduled property, when TRUE, specifies that the link is scheduled for periodic activation, as compared with asynchronous, on-demand activation. The StateScheduled property corresponds to the fourth bit (0x00000008) of the StateFlags property.
SupportedLinkActions Property	The SupportedLinkActions property specifies the actions supported by the link. The individual bits of this property are available as the Action... properties in this class.
TypeCurrentlyUnreachable Property	The TypeCurrentlyUnreachable property, when TRUE, specifies that the link holds messages for destinations that currently cannot be reached. The TypeCurrentlyUnreachable property corresponds to the thirteenth bit (0x00001000) of the StateFlags property.
TypeDeferredDelivery Property	The TypeDeferredDelivery property, when TRUE, specifies that the link holds mail that is awaiting a trigger to start transmission. The TypeDeferredDelivery property corresponds to the fourteenth bit (0x00002000) of the StateFlags property.
TypeInternal Property	The TypeInternal property indicates that the link is used for internal message processing. The TypeInternal property corresponds to the fifteenth bit (0x00004000) of the StateFlags property.
TypeLocalDelivery Property	The TypeLocalDelivery property, when TRUE, specifies that the link handles local mail delivery. The TypeLocalDelivery property corresponds to the tenth bit (0x00000200) of the StateFlags property.
TypePendingCategorization Property	The TypePendingCategorization property, when TRUE, specifies that the link is resolving addresses against entries in Active Directory. The TypePendingCategorization property corresponds to the twelfth bit (0x00000800) of the StateFlags property.
TypePendingRouting Property	The TypePendingRouting property, when TRUE, specifies that the link is determining the routing of the next message that is waiting to be transmitted. The TypePendingRouting property corresponds to the eleventh bit (0x00000400) of the StateFlags property.
TypePendingSubmission Property	The TypePendingSubmission property, when TRUE, specifies that the link handles messages that have not yet been submitted to the routing engine. The TypePendingSubmission property corresponds to the sixteenth bit (0x00008000) of the StateFlags property.
TypeRemoteDelivery Property	The TypeRemoteDelivery property, when TRUE, specifies that the link is currently handling a remote message delivery. The TypeRemoteDelivery property corresponds to the ninth bit (0x00000100) of the StateFlags property.
Version Property	The Version property specifies the version number of the underlying link control software.
VirtualMachine Property	The VirtualMachine property specifies the name of the virtual machine that is the source of the link.
VirtualServerName Property	The value of the VirtualServerName property is the integer number of the virtual machine that is the source of the link. This number is the Microsoft Active Directory® common name (CN) for the virtual server object.

Table 5.22 *The ExchangeQueue Properties*

Name	Description
CanEnumAll Property	The CanEnumAll property, when TRUE, specifies that the queue can enumerate all of the messages that it has waiting for transmission. The CanEnumAll property corresponds to the thirty-first bit (0x40000000) of the MsgEnumFlagsSupported property.
CanEnumFailed Property	The CanEnumFailed property, when TRUE, specifies that the queue can enumerate the messages that it has waiting for transmission that it was unable to transfer. The CanEnumFailed property corresponds to the ninth bit (0x00000100) of the MsgEnumFlagsSupported property.
CanEnumFirstNMessages Property	The CanEnumFirstNMessages property, when TRUE, specifies that the queue can enumerate the first N messages that it has waiting for transmission. The CanEnumFirstNMessages property corresponds to the first bit (0x00000001) of the MsgEnumFlagsSupported property.
CanEnumFrozen Property	The CanEnumFrozen property, when TRUE, specifies that the queue can enumerate messages that it has waiting for transmission that have been frozen. The CanEnumFrozen property corresponds to the sixth bit (0x00000020) of the MsgEnumFlagsSupported property.
CanEnumInvertSense Property	The CanEnumInvertSense property, when TRUE, specifies that the queue can enumerate messages that it has waiting for transmission that do not match the criteria requested. For example, requesting the oldest messages while inverting the request sense would return the newest messages. The CanEnumInvertSense property corresponds to the thirty-second bit (0x80000000) of the MsgEnumFlagsSupported property.
CanEnumLargerThan Property	The CanEnumLargerThan property, when TRUE, specifies that the queue can enumerate the messages that it has waiting for transmission that are larger than a specified value. The CanEnumLargerThan property corresponds to the fourth bit (0x00000008) of the MsgEnumFlagsSupported property.
CanEnumNLargestMessages Property	The CanEnumNLargestMessages property, when TRUE, specifies that the queue can enumerate the specified number of the largest messages that it has waiting for transmission. The CanEnumNLargestMessages property corresponds to the seventh bit (0x00000040) of the MsgEnumFlagsSupported property.
CanEnumNOldestMessages Property	The CanEnumNOldestMessages property, when TRUE, specifies that the queue can enumerate the specified number of the oldest messages that it has waiting for transmission. The CanEnumNOldestMessages property corresponds to the eighth bit (0x00000080) of the MsgEnumFlagsSupported property.
CanEnumOlderThan Property	The CanEnumOlderThan property, when TRUE, specifies that the queue can enumerate the messages that it has waiting for transmission that arrived before a specified date and time. The CanEnumOlderThan property corresponds to the fifth bit (0x00000010) of the MsgEnumFlagsSupported property.
CanEnumRecipient Property	The CanEnumRecipient property, when TRUE, specifies that the queue can enumerate the recipients of messages that it has waiting for transmission. The CanEnumRecipient property corresponds to the fourth bit (0x00000004) of the MsgEnumFlagsSupported property.
CanEnumSender Property	The CanEnumSender property, when TRUE, specifies that the queue can enumerate the senders of messages that it has waiting for transmission. The CanEnumSender property corresponds to the second bit (0x00000002) of the MsgEnumFlagsSupported property.
GlobalStop Property	The GlobalStop property specifies whether the queue is currently stopped.
IncreasingTime Property	The IncreasingTime property specifies the amount of time, in milliseconds, that the number of messages waiting to be transferred by the queue has been increasing.
LinkName Property	The LinkName property specifies the name of the link in which this queue is contained.
MsgEnumFlagsSupported Property	The MsgEnumFlagsSupported property specifies a bit-mapped set of flags that indicate what types of objects can be enumerated. The individual bits of this property are available as the queue CanEnum... properties in th class.
NumberOfMessages Property	The NumberOfMessages property specifies the number of messages that are waiting for transmission by the queue.
ProtocolName Property	The ProtocolName property specifies the transmission protocol for the queue.
QueueName Property	The QueueName property specifies the name of the queue.
SizeOfQueue Property	The SizeOfQueue property specifies the total size of all messages in the queue, in bytes.
Version Property	The Version property specifies the version number of the Exchange software.
VirtualMachine Property	The VirtualMachine property specifies the name of the virtual machine that is the source of the link.
VirtualServerName Property	The value of the VirtualServerName property is the integer number of the virtual machine that is the source of the queue. This number is the Microsoft Active Directory common name (CN) for the virtual server object.

Table 5.23 *The ExchangeClusterResource Properties*

Name	Description
Name Property	The Name property returns the name of the Exchange cluster resource.
Owner Property	The Owner property for a cluster resource specifies the cluster node of which the resource is a part.
State Property	The State property specifies the current state of the cluster resource.
Type Property	The Type property specifies the resource type.
VirtualMachine Property	The VirtualMachine property returns the name of the virtual machine that owns this resource.

Figure 5.26 *The "Message Tracking" user interface.*

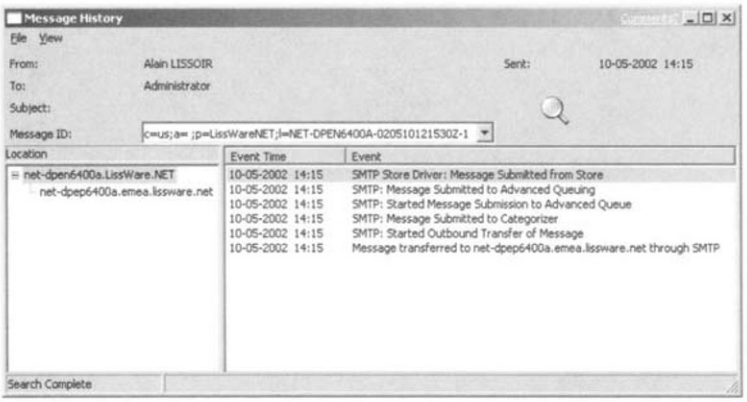

Table 5.24 *The Exchange_MessageTrackingEntry Class Properties*

Name	Description
KeyID	The KeyID property uniquely identifies the message to which the log entry pertains.
AttemptedPartnerServer	The AttemptedPartnerServer property indicates the server to which Exchange tried to send a message, but was unable to complete the transfer.
ClientIP	The ClientIP property indicates the Transmission Control Protocol/Internet Protocol (TCP/IP) address of the messaging client that originally submitted the message.
ClientName	The ClientName property indicates the name of the messaging client application that submitted the message.
Cost	The Cost property indicates the relative effort required to transfer the message. There are no specific units used in this property. Higher values indicate slower network connections must be used, or a greater number of transfers is required to transfer the message.
DeliveryTime	The DeliveryTime property indicates the date and time, in coordinated universal time (UTC), when the message was transferred successfully from the computer running Microsoft Exchange 2000 Server.
Encrypted	The Encrypted property indicates, when TRUE, that the message is encrypted.
EntryType	The EntryType property indicates what occurred to cause the message tracking log entry to be created.
ExpansionDL	The ExpansionDL property indicates the name of the Exchange distribution list that was expanded. After the distribution list is expanded, the message recipient list includes the names of the individual members of that distribution list.
LinkedMessageID	The LinkedMessageID property provides a string that can be used to retrieve message tracking log entries for the message after it has been transferred.
MessageID	The MessageID property indicates the identifier string for the message. The identifier may be assigned by the messaging client application or by the computer running Exchange 2000 Server.
OriginationTime	The OriginationTime property indicates the date and time, in UTC, when the message was created by the messaging client application.
PartnerServer	The PartnerServer property indicates the server to which Exchange transferred the message.
Priority Property	The Priority property specifies the importance of the message, as displayed by the messaging client application. "Urgent" "Normal" "Not Urgent"
RecipientAddress	The RecipientAddress property is an array that specifies the email addresses of each message recipient.
RecipientCount	The RecipientCount property indicates how many recipients are in the recipients list for the message.
RecipientStatus	The RecipientStatus property is an array that indicates the status of an individual message recipient.
SenderAddress	The SenderAddress property specifies the email address of the message sender.
ServerIP	The ServerIP property indicates the TCP/IP protocol address of the computer running Exchange 2000 Server.
ServerName	The Server property indicates the computer name of the computer running Exchange 2000 Server that created the message tracking log entry.
Size	The Size property indicates the message size, including attachments, in bytes.
Subject	The Subject property indicates the subject of the message, as found in the Subject: message header.
SubjectID	The SubjectID property specifies an identifier created by the messaging client application.
TimeLogged	The TimeLogged property indicates the date and time, in UTC, when the message tracking log entry was created.
Version	The Version property indicates the version of the service that created the message tracking log entry.

Table 5.25 *The Exchange_MessageTrackingEntry EntryType Property Meaning*

Values	Description
0	Message received through X400
1	tevtProbeTransferIn
3	Report received
4	Message submitted
5	tevtProbeSubmission
6	tevtProbeTransferOut
7	Message transferred out
8	Report transferred out
9	Message delivered
10	Report delivered
18	tevtStartAssocByMTSUser
23	tevtReleaseAssocByMTSUser
26	Distribution list expanded
28	Message redirected
29	Message rerouted
31	Server downgraded by MTA
33	Report absorbed
34	Report generated
43	Unroutable report discarded
50	Message deleted by Administrator
51	Probe deleted by Administrator
52	Report deleted by Administrator
1000	Message delivered locally
1001	Message transferred in over backbone
1002	Message transferred out over backbone
1003	Message transferred out over gateway
1004	Message transferred in over gateway
1005	Report transferred in over gateway
1006	Report transferred out over gateway
1007	Report generated
1010	SMTP: Message queued outbound
1011	SMTP: Message transferred out
1012	SMTP: Inbound message received
1013	SMTP: Inbound message transferred
1014	SMTP: Message rerouted
1015	SMTP: Report transferred in
1016	SMTP: Report transferred out
1017	SMTP: Report generated
1018	SMTP: Report absorbed
1019	SMTP: Message submitted to Advanced Queuing
1020	SMTP: Started outbound transfer of message
1021	SMTP: Message sent to badmail directory
1022	SMTP: Advanced Queue failure
1023	SMTP: Message delivered locally
1024	SMTP: Message submitted to Categorizer
1025	SMTP: Started message submission to Advanced Queue
1026	SMTP: Advanced Queue failed to deliver message
1027	SMTP Store Driver: Message submitted from Store
1028	SMTP Store Driver: Message delivered locally to Store
1029	SMTP Store Driver: Message submitted to MTA
1030	SMTP: Non-delivery report (NDR) generated
1031	SMTP: Message transferred out

the **GetCollectionOfInstances.wsf** script (see Sample 1.5, "Listing all instances of a class with their properties formatted"), it is possible to list all instances of the *Exchange_MessageTrackingEntry* class.

```
1:  C:\>GetCollectionOfInstances.wsf Exchange_MessageTrackingEntry
                /NameSpace:Root\MicrosoftExchangeV2 /Machine:net-dpep6400a.emea.LissWare.NET
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:  ClientIP: ............................... 10.10.10.3
6:  ClientName: ............................. net-dpen6400a.LissWare.NET
7:  DeliveryTime: ........................... 0
8:  Encrypted: .............................. FALSE
```

```
 9:   EntryType: ............................. 1019
10:   InstallDate: ............................ 10-05-2002 12:15:01
11:   *KeyID: ..............................
                    \\net-dpep6400a.Emea.LissWare.NET\NET-DPEP6400A.log\20020510.log,2082
12:   MessageID: ..............................
                    8C05225BE6E2A5438AF2FC417C0D60BB2C8F@net-dpen6400a.LissWare.NET
13:   OriginationTime: ........................ 10-05-2002 12:15:31
14:   Priority: ................................ 0
15:   RecipientAddress: ........................ Administrator@LissWare.NET
16:   RecipientCount: .......................... 1
17:   RecipientStatus: ......................... 0
18:   SenderAddress: ........................... Alain.Lissoir@LissWare.NET
19:   ServerIP: ................................ 192.10.10.3
20:   ServerName: .............................. net-dpep6400a.Emea.LissWare.NET
21:   Size: .................................... 3339
22:   TimeLogged: .............................. 10-05-2002 12:15:31
23:   Version: ................................. Version: 6.0.3590.0
24:
25:   ClientIP: ................................ 10.10.10.3
26:   ClientName: .............................. net-dpen6400a.LissWare.NET
27:   DeliveryTime: ............................ 0
28:   Encrypted: ............................... FALSE
29:   EntryType: ............................... 1025
30:   InstallDate: ............................. 10-05-2002 12:15:31
31:   *KeyID: ..............................
                    \\net-dpep6400a.Emea.LissWare.NET\NET-DPEP6400A.log\20020510.log,2361
32:   MessageID: ..............................
                    8C05225BE6E2A5438AF2FC417C0D60BB2C8F@net-dpen6400a.LissWare.NET
33:   OriginationTime: ........................ 10-05-2002 12:15:31
34:   Priority: ................................ 0
35:   RecipientAddress: ........................ Administrator@LissWare.NET
36:   RecipientCount: .......................... 1
37:   RecipientStatus: ......................... 0
38:   SenderAddress: ........................... Alain.Lissoir@LissWare.NET
39:   ServerIP: ................................ 192.10.10.3
40:   ServerName: .............................. net-dpep6400a.Emea.LissWare.NET
41:   Size: .................................... 3339
42:   TimeLogged: .............................. 10-05-2002 12:15:31
43:   Version: ................................. Version: 6.0.3590.0
44:
..:
84:
85:   DeliveryTime: ............................ 1
86:   Encrypted: ............................... FALSE
87:   EntryType: ............................... 1028
88:   InstallDate: ............................. 10-05-2002 12:15:31
89:   *KeyID: ..............................
                    \\net-dpep6400a.Emea.LissWare.NET\NET-DPEP6400A.log\20020510.log,3198
90:   MessageID: ..............................
                    8C05225BE6E2A5438AF2FC417C0D60BB2C8F@net-dpen6400a.LissWare.NET
91:   OriginationTime: ........................ 10-05-2002 12:15:31
92:   Priority: ................................ 0
93:   RecipientAddress: ........................ Administrator@LissWare.NET
94:   RecipientCount: .......................... 1
95:   RecipientStatus: ......................... 0
96:   SenderAddress: ........................... Alain.Lissoir@LissWare.NET
97:   ServerName: .............................. net-dpep6400a.Emea.LissWare.NET
98:   Size: .................................... 3339
99:   TimeLogged: .............................. 10-05-2002 12:15:32
```

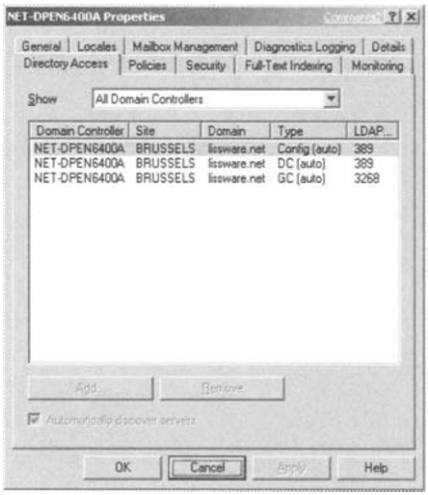

Figure 5.27
The "DSAccess" user interface.

5.3.2.5 The DSAccess provider

The *DSAccess* component of Exchange 2000 manages access to the Active Directory for various Exchange components. *DSAccess* also caches Active Directory requests to improve performance. Because the Active Directory is a critical component of Exchange 2000, it is important to closely monitor which Active Directory server(s) Exchange 2000 is using. Having too many Exchange 2000 servers working with the same Active Directory server(s) may seriously impact overall system performance. As shown in Figure 5.27, once Service Pack 2 is installed, the Exchange System Manager displays which Active Directory Domain Controller is being used by the Exchange 2000 server.

Besides this user interface extension, Microsoft also implemented a WMI provider, called *ExchangeDsAccessProvider*, supporting one WMI class called the *Exchange_DSAccessDC* class. Basically, this provider, with its supported class, retrieves the same set of information. However, due to the WMI capabilities, it is possible to monitor the *DSAccess* state—something that is not possible from the Exchange System Manager. By reusing the **GetCollectionOfInstances.wsf** script (see Sample 1.5, "Listing all instances of a class with their properties formatted"), the following output shows the list of properties exposed by the *Exchange_DSAccessDC* class.

```
1:  C:\>GetCollectionOfInstances.wsf Exchange_DSAccessDC /NameSpace:Root\MicrosoftExchangeV2
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
```

```
 6:   ConfigurationType: ..... 1
 7:   DirectoryType: ..... 0
 8:   InstallDate: ..... 01-01-2000
 9:   IsFast: ..... TRUE
10:   IsInSync: ..... TRUE
11:   IsUp: ..... TRUE
12:   LDAPPort: ..... 389
13:   *Name: ..... net-dpen6400a.LissWare.NET
14:   *Type: ..... 0
15:
16:   ConfigurationType: ..... 1
17:   DirectoryType: ..... 0
18:   InstallDate: ..... 01-01-2000
19:   IsFast: ..... TRUE
20:   IsInSync: ..... TRUE
21:   IsUp: ..... TRUE
22:   LDAPPort: ..... 389
23:   *Name: ..... net-dpen6400a.LissWare.NET
24:   *Type: ..... 1
25:
26:   ConfigurationType: ..... 1
27:   DirectoryType: ..... 0
28:   InstallDate: ..... 01-01-2000
29:   IsFast: ..... TRUE
30:   IsInSync: ..... TRUE
31:   IsUp: ..... TRUE
32:   LDAPPort: ..... 3268
33:   *Name: ..... net-dpen6400a.LissWare.NET
34:   *Type: ..... 2
```

Table 5.26 lists the *Exchange_DSAccessDC* properties.

Table 5.26 *The Exchange_DSAccessDC Class Properties*

Name	Description
Name	The Name property specifies the computer name of the domain controller. When creating new DSAccessDC instances, or when retrieving a specific DSAccessDC instance, the Name property is required.
Type	The DirectoryType property indicates the role that the domain controller plays in the Exchange system. "Configuration Domain Controller" "Local Domain Controller" "Global Catalog"
AsyncConnectionCount	The AsyncConnectionCount indicates the number of asynchronous (non-blocking) connections currently open between the computer running Exchange 2000 Server and the domain controller.
ConfigurationType	The ConfigurationType property indicates whether the instance describes a domain controller that was detected automatically, or one that was specified manually. "Manual" "Automatic"
DirectoryType	The DirectoryType property indicates whether the domain controller is an Active Directory or an Exchange Server 5.5 directory service. "Active Directory" "Exchange 5.5 Directory"
IsFast	The IsFast property indicates, when TRUE, that the domain controller response time has been less than two seconds.
IsInSync	The IsInSync property indicates whether the domain controller is synchronized with the Global Catalog server and with the Configuration domain controller.
IsUp	The IsUp property indicates whether the domain controller was available the last time Exchange attempted to access it.
LDAPPort	The LDAPPort property specifies the Transmission Control Protocol/Internet Protocol (TCP/IP) port on which the domain controller listens for Lightweight Directory Access Protocol (LDAP) requests.
SyncConnectionCount	The SyncConnectionCount indicates the number of synchronous (blocking) connections currently open between the computer running Exchange 2000 Server and the domain controller.

With the script in Sample 6.17 ("A generic script for asynchronous event notification") in the appendix, the following command-line sample can be used to receive a notification for any changes that occur to any instances of the *Exchange_DSAccessDC* class. Note the presence of the **WITHIN** statement, since none of the Exchange WMI providers is implemented as event providers. The WQL event query will look as follows:

```
Select * From __InstanceModificationEvent Within 5 Where TargetInstance ISA 'Exchange_DSAccessDC'
```

5.3.3 SQL Server 2000

When installing SQL Server 2000, no WMI support is provided with the installation. To add the WMI management capabilities to your SQL Server installation, another setup from the folder "\x86\Other\WMI," located on the SQL Server 2000 CD, must be executed. This setup creates the **Root\MicrosoftSQLServer** namespace and registers the unique WMI SQL provider implemented as an instance and method provider in the CIM repository (see Table 5.27).

During the setup, a collection of WMI classes is created in the **Root\MicrosoftSQLServer** namespace that maps manageable instances of SQL Server 2000. The SQL Server WMI classes model objects, such as databases and tables, with their associated settings. The SQL Server WMI implementation provides several management capabilities, including:

- Create, change, or delete managed objects, such as creating or deleting a database. For instance, to create a database, the *Create* method of the *MSSQL_Database* class must be invoked. However, to delete a database, the WMI instance of the database must be retrieved, and then the *Delete_* method of the **SWBemObject** object representing the database instances must be invoked.
- Administer managed objects, such as backing up databases and logs. For instance, invoking the *SQLBackup* method of the *MSSQL_SQLServer* class will perform a backup of the SQL server.
- Retrieve information on specific managed objects, such as determining whether full-text indexing is enabled on a table. This can be done easily by reading the *FullTextIndexActive* property of the *MSSQL_Table* instance.
- Query managed objects that meet a specific criterion, such as listing all encrypted stored procedures. This can be performed with some traditional WQL data queries.

Table 5.27 *The WMI SQL Provider Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete
MS SQL Provider											
MicrosoftSQLServer 2000	Root/MicrosoftSQLServer		X	X				X	X	X	X

- Execute methods defined for managed objects. For instance, to rebuild indexes from a table it is necessary to invoke the *RebuildIndex* method of the *MSSQL_Table* class.
- Generate events when a managed object is created, changed, or deleted, such as sending an event when a database option is changed. This can be done easily with a WQL event query. However, since the SQL WMI provider is not implemented as an event provider, the WQL event query requires the use of the **WITHIN** statement.
- Enumerate managed objects. As another example, to list all tables in a database it is necessary to retrieve all *MSSQL_Table* instances associated with the *MSSQL_Database* instance.
- Describe relationships between managed objects. For instance, to identify which logins are authorized to access a database it is necessary to enumerate *MSSQL_Login* instances associated with the *MSSQL_Database* instance via the *MSSQL_DatabaseLogin* association class.

Besides these functionalities, there are two interesting points to note:

- The SQL Server 2000 WMI implementation maps over the SQL Distributed Management Objects API (SQL-DMO) but does not support the management of the SQL replication.
- The SQL Server 2000 WMI implementation can also be used with SQL Server 7.0.

The SQL WMI classes supported are listed in Table 5.28.

It is important to mention that the **Root\MicrosoftSQLServer** namespace doesn't only contain SQL server classes. It also contains classes from the **Root\CIMv2** namespace. Actually, the **Root\MicrosoftSQLServer** namespace contains a registration of the WMI *View* provider (see Chapter 3, section 3.9.1) to make classes such as *Win32_Service* and *Win32_Process*

Table 5.28 *The WMI SQL Classes*

Name	Type	Description
MSSQL_BackupDevice	Dynamic	The MSSQL_BackupDevice class represents backup devices known to the SQL Server installation.
MSSQL_Check	Dynamic	The MSSQL_Check class represents the attributes of a SQL Server integrity constraint.
MSSQL_Column	Dynamic	The MSSQL_Column class represents columns in a SQL Server table.
MSSQL_ConfigValue	Dynamic	The MSSQL_ConfigValue class represents the SQL Server configuration values. Some SQL Server configuration options do not take effect until the SQL Server service has been stopped and restarted. You can force the server to immediately accept changes in some options by using the ReconfigureWithOverride method. The DynamicReconfigure property indicates whether the ConfigValue object requires a restart.
MSSQL_Database	Dynamic	The MSSQL_Database class represents a SQL Server database. Each SQL Server installation can contain one or more databases.
MSSQL_DatabaseFile	Dynamic	The MSSQL_DatabaseFile class is an extension to the CIM_DataFile class. It contains properties that are relevant to an operating system file that is also a file storing SQL Server database data.
MSSQL_DatabaseRole	Dynamic	The DatabaseRole object represents the properties of a SQL Server database role. SQL Server database roles establish groups of users with similar security attributes. Database permissions can be granted by role, simplifying database security planning and administration.
MSSQL_DatabaseSetting	Dynamic	The MSSQL_DatabaseSetting class represents the settings for a database. These settings control the access to and the behavior of the database.
MSSQL_Default	Dynamic	The MSSQL_Default object represents the attributes of a SQL Server default. Defaults provide data to columns and user-defined data types when no other data is available on an INSERT statement execution. Unlike DRI defaults, represented by the MSSQL_DRIDefault class, the defaults represented by the MSSQL_Default class can be bound to one or more columns and datatypes.
MSSQL_DRIDefault	Dynamic	The MSSQL_DRIDefault class represents the properties of a Microsoft SQL Server column default. Defaults provide data to columns and user-defined data types when no other data is available on an INSERT statement execution. Unlike other defaults, the Declarative Referential Integrity defaults are associated with one and only one column.
MSSQL_ErrorLog	Dynamic	The MSSQL_ErrorLog class represents the error logs used by the SQL Server installation.
MSSQL_ErrorLogEntry	Dynamic	The MSSQL_ErrorLogEntry class represents the entries in a SQL Service error log.
MSSQL_FileGroup	Dynamic	The MSSQL_FileGroup class represents the groups of operating system files used to store a database. A SQL Server filegroup categorizes the operating system files containing data from a single SQL Server database to simplify database administration tasks, such as backup. A filegroup cannot contain the operating system files of more than one database, though a single database can contain more than one filegroup.
MSSQL_ForeignKey	Dynamic	The MSSQL_ForeignKey class represents the foreign keys defined for a SQL Server database table.
MSSQL_FullTextCatalog	Dynamic	The MSSQL_FullTextCatalog class represents a single Microsoft Search persistent data store. Microsoft Search enables full-text queries on data maintained by Microsoft SQL Server. The service both builds the indexes providing full-text query capability and participates in query resolution by providing result data during a full-text query. Index data is maintained within a full-text catalog.
MSSQL_FullTextCatalogService	Dynamic	The MSSQL_FullTextCatalogService class represents the Microsoft Search full-text indexing service. The Microsoft Search full-text indexing service enables full-text queries on data maintained by SQL Server. Microsoft Search both builds the indexes providing full-text query capability and participates in query resolution by providing result data during a full-text query.
MSSQL_Index	Dynamic	The MSSQL_Index class represents an index for a SQL Server table. A SQL Server index optimizes access to data in SQL Server tables. Indexes are also used to enforce some constraints, such as UNIQUE and PRIMARY KEY constraints.
MSSQL_IndexTableInformation	Dynamic	The MSSQL_IndexTableInformation class represents the information regarding the age and structure of the index statistical information.
MSSQL_IntegratedSecuritySetting	Dynamic	The MSSQL_IntegratedSecuritySetting class represents the security settings of a SQL Server installation. This setting affects all login connections to SQL Server regardless of the login authentication type.
MSSQL_LanguageSetting	Dynamic	The MSSQL_LanguageSetting class represents the properties of an installed SQL Server language record. Language record identifiers categorize system messages so that error and status information can be presented as localized text. A language record specifies the format for dates displayed in system messages.
MSSQL_Login	Dynamic	The MSSQL_Login class represents the login authentication records present in a SQL Server installation.
MSSQL_PrimaryKey	Dynamic	The MSSQL_PrimaryKey class represents a primary key of a table. A primary key must also be a candidate key of the table.
MSSQL_Process	Dynamic	The MSSQL_Process class represents SQL Server processes. Note that these are not the same as an operating system's notion of a process. These are the processes identified by the SQL Server and assigned a SQL Server process ID by SQL Server.
MSSQL_RegistrySetting	Dynamic	The MSSQL_RegistrySetting class represents the installation and run-time parameters of SQL Server stored in the registry.
MSSQL_Rule	Dynamic	The MSSQL_Rule class represents a single Microsoft SQL Server data-integrity rule. SQL Server offers several mechanisms for ensuring data integrity. A SQL Server rule is a Transact-SQL condition_expression syntax element that defines a data-integrity constraint. A rule can be bound to a column or user-defined data type.

Table 5.28 *The WMI SQL Classes (continued)*

Name	Type	Description
MSSQL_SQLServer	Dynamic	The MSSQL_SQLServer class represents a SQL Server installation. There will be one instance of this class for each installation of SQL Server on the computer system.
MSSQL_SQLServerRole	Dynamic	The MSSQL_SQLServerRole class represents a SQL Server security role not constrained to operation within a single database. Roles are used to establish groups of users, in order to make it convenient to set permissions for a group of users.
MSSQL_StoredProcedure	Dynamic	The MSSQL_StoredProcedure class represents standard as well as extended stored procedure defined in a SQL Server database. SQL Server stored procedures can contain input and output parameters and can return the results of one or more SELECT statements or a single long integer. In order to create an instance of a new stored procedure, the Text properties need to be specified along with the key properties of the class. The Text property specifies the Transact-SQL script that defines the stored procedure.
MSSQL_StoredProcedureParameter	Dynamic	The MSSQL_StoredProcedureParameter class represents the input and output parameters of a SQL Server stored procedure.
MSSQL_SystemDatatype	Dynamic	The MSSQL_SystemDatatype class represents base data type defined in Microsoft SQL Server.
MSSQL_Table	Dynamic	The MSSQL_Table class represents a table in the SQL Server database.
MSSQL_TransactionLog	Dynamic	The MSSQL_TransactionLog class represents the transaction log of a Microsoft SQL Server database. A SQL Server transaction log maintains a record of modifications to the Operating System files containing the data of an SQL Server database. The transaction log provides data-recovery assistance in the event of system failure and an SQL Server database has at least one operating system file that stores transaction log records. A transaction log can be written to more than one operating system file. Each SQL Server database maintains its own transaction log and the operating system file or files that store log records cannot be shared with another database.
MSSQL_Trigger	Dynamic	The MSSQL_Trigger class represents a trigger. SQL Server supports using triggers as a kind of stored procedure. Triggers are executed when a specified data modification, such as an attempt to delete a row, is attempted on the table on which the trigger is defined.
MSSQL_UniqueKey	Dynamic	The MSSQL_UniqueKey object represents a unique key in a database. All candidate keys that are not the primary key are unique keys.
MSSQL_User	Dynamic	The User object exposes the attributes of a single Microsoft SQL Server database user.
MSSQL_UserDatatype	Dynamic	The MSSQL_UserDatatype class represents a data type defined by a user.
MSSQL_UserDefinedFunction	Dynamic	The MSSQL_UserDefinedFunction class represents a user-defined function in the SQL Server database.
MSSQL_View	Dynamic	The MSSQL_View class represents view tables in the database.
MSSQL_BaseDatatype	Association	The MSSQL_BaseDatatype class represents an association between a user-defined datatype and the system datatype from which it is derived.
MSSQL_ColumnDatatype	Association	The MSSQL_ColumnDatatype class associates a column its data type.
MSSQL_ColumnDefault	Association	The MSSQL_ColumnDefault class associates a column to the default for the column.
MSSQL_ColumnDRIDefault	Association	The MSSQL_ColumnDRIDefault class associates a column to a DRI default.
MSSQL_ColumnRule	Association	The MSSQL_ColumnRule class represents an association between a column and a rule bound to the column.
MSSQL_DatabaseCandidateKey	Association	The MSSQL_DatabaseCandidateKey class represents an association between a database and a candidate key that is present in one of the tables in the database. This association allows an application to perform a single traversal to find the candidate keys in a database
MSSQL_DatabaseDatabaseRole	Association	The MSSQL_DatabaseDatabaseRole class associates database role to the database within which the role is defined.
MSSQL_DatabaseDatabaseSetting	Association	The MSSQL_DatabaseDatabaseSetting class associates a SQL Server database to an instance of the MSSQL_DatabaseSetting class that contains the settings for the database.
MSSQL_DatabaseDatatype	Association	The MSSQL_DatabaseDatatype class associates a database to the datatypes defined within the database.
MSSQL_DatabaseDefault	Association	The MSSQL_DatabaseDefault association associates a database to the defaults defined within the database.
MSSQL_DatabaseFileDataFile	Association	The MSSQL_DatabaseFileDataFile class associates a CIM_DataFile class to the MSSQL_DatabaseFile class that contains database file-specific properties of an Operating System file.
MSSQL_DatabaseFileGroup	Association	The MSSQL_DatabaseFileGroup class represents an association between a database and the file group that contains the operating system files that store the data for the database.
MSSQL_DatabaseFullTextCatalog	Association	The MSSQL_DatabaseFullTextCatalog class represents an association between a database and a full-text catalog that stores index data used for full-text queries against the database.
MSSQL_DatabaseLogin	Association	The MSSQL_DatabaseLogin class represents an association between a database and a login that is mapped to a user defined in the database. This association allows an application to perform a single traversal to find the logins mapped to the users defined in the database.
MSSQL_DatabaseOwnerLogin	Association	The MSSQL_DatabaseOwnerLogin class represents an association between a database and the login mapped to the user that owns the database.
MSSQL_DatabaseRoleDatabasePermission	Association	The MSSQL_DatabaseRoleDatabasePermission class represents the permissions that a database role has for the database in which it is defined. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a database role has permissions to access a database by virtue of being a member of another database role, then there will not be a permission association instance between the role and the database.

Table 5.28 *The WMI SQL Classes (continued)*

Name	Type	Description
MSSQL_DatabaseRoleStoredProcedurePermission	Association	The MSSQL_DatabaseRoleStoredProcedurePermission class represents the permissions that a database role has for a stored procedure. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a database role has permissions to access the stored procedure by virtue of being a member of another database role, then there will not be a permission association instance between the role and the stored procedure.
MSSQL_DatabaseRoleTablePermission	Association	The MSSQL_DatabaseRoleTablePermission class represents the permissions that a database role has for a table. The instances of this class represent only the permissions that have been explicitly granted or denied to the user object. For example, if a database role has permissions to access the table by virtue of being a member of another database role, then there will not be a permission association instance between the role and the table.
MSSQL_DatabaseRoleUserDefinedFunctionPermission	Association	The MSSQL_DatabaseRoleUserDefinedFunctionPermission class represents the permissions that a database role has for a table. The instances of this class represent only the permissions that have been explicitly granted or denied to the user object. For example, if a database role has permissions to access the user-defined function by virtue of being a member of another database role, then there will not be a permission association instance between the role and the user-defined function.
MSSQL_DatabaseRoleViewPermission	Association	The MSSQL_DatabaseRoleViewPermission class represents the permissions that a database role has for a view. The instances of this class represent only the permissions that have been explicitly granted or denied to the user object. For example, if a database role has permissions to access the view by virtue of being a member of another database role, then there will not be a permission association instance between the role and the view.
MSSQL_DatabaseRule	Association	The MSSQL_DatabaseRule class represents an association between a database and the rules defined within the database.
MSSQL_DatabaseStoredProcedure	Association	The MSSQL_DatabaseStoredProcedure class represents an association between the database and a stored procedure defined within the database.
MSSQL_DatabaseTable	Association	The MSSQL_DatabaseTable class represents an association between a database and a table contained within the database.
MSSQL_DatabaseTransactionLog	Association	The MSSQL_DatabaseTransactionLog class represents an association between the database and the transaction log for the database.
MSSQL_DatabaseUser	Association	The MSSQL_DatabaseUser class represents an association between a database and a user defined for the database.
MSSQL_DatabaseUserDefinedFunction	Association	The MSSQL_DatabaseUserDefinedFunction class represents an association between a database and a user-defined function defined within the database.
MSSQL_DatabaseView	Association	The MSSQL_DatabaseView class represents an association between a database and a view contained within the database.
MSSQL_DBMSObjectOwner	Association	The MSSQL_DBMSObjectOwner class represents an association between a SQL database object and the user who owns the object.
MSSQL_ErrorLogDataFile	Association	The MSSQL_ErrorLogDataFile class represents an association between a SQL Server error log, and the operating system file used to store the error log.
MSSQL_ErrorLogErrorLogEntry	Association	The MSSQL_ErrorLogErrorLogEntry class represents an association between an error log and an entry in the error log.
MSSQL_FileGroupDatabaseFile	Association	The MSSQL_FileGroupDatabaseFile class represents an association between a database file group and an operating system files that is part of the group.
MSSQL_FullTextWin32Service	Association	The MSSQL_FullTextWin32Service represents an association between an instance of MSSQL_FullTextCatalogService and the corresponding instance of the Win32_Service.
MSSQL_IndexColumn	Association	The MSSQL_IndexColumn class represents an association between an index and a column that participates in the index.
MSSQL_IndexFileGroup	Association	The MSSQL_IndexFileGroup class represents an association between an index and a file group that stores the index.
MSSQL_IndexStatistics	Association	The MSSQL_IndexStatistics class represents an association between an index and the statistical information stored with the index.
MSSQL_KeyColumn	Association	The MSSQL_KeyColumn class represents an association between a key and a column that is part of the key.
MSSQL_KeyFileGroup	Association	The MSSQL_KeyFileGroup class represents an association between a key and the file group used to store the key.
MSSQL_LoginDefaultDatabase	Association	The MSSQL_LoginDefaultDatabase class represents an association between a login and the default database for the login.
MSSQL_LoginWin32Group	Association	The MSSQL_LoginWin32Group class represents an association between a login and the Win32 user group used for authentication by the login.
MSSQL_LoginWin32UserAccount	Association	The MSSQL_LoginWin32UserAccount class represents an association between a login and the Win32 user account used for authentication by the login.
MSSQL_MemberDatabaseRole	Association	The MSSQL_MemberDatabaseRole class represents an association between two database roles, one being a member of the other.
MSSQL_MemberLogin	Association	The MSSQL_MemberLogin class represents an association between a SQL Server role and a login that is a member of the role.
MSSQL_MemberUser	Association	The MSSQL_MemberUser class represents an association between a database role and a user that is a member of the role.
MSSQL_ReferencedKey	Association	The MSSQL_ReferencedKey class represents an association between a foreign key and the candidate key that the foreign key references.
MSSQL_ReferencedTable	Association	The MSSQL_ReferencedTable class represents an association between a foreign key and the table that contains the primary key referenced by the foreign key.
MSSQL_SQLServerBackupDevice	Association	The MSSQL_SQLServerBackupDevice class represents an association between a SQL Server installation and a backup device known to SQL Server.

Table 5.28 *The WMI SQL Classes (continued)*

Name	Type	Description
MSSQL_SQLServerConfigValue	Association	The MSSQL_SQLServerConfigValue class represents an association between a SQL Server installation and the configured value settings for the installation.
MSSQL_SQLServerDatabase	Association	The MSSQL_SQLServerDatabase class represents an association between a SQL Server installation and a database that is part of the installation.
MSSQL_SQLServerErrorLog	Association	The MSSQL_SQLServerErrorLog represents an association between a SQL Server installation and the error log used by the installation.
MSSQL_SQLServerIntegratedSecuritySetting	Association	The MSSQL_SQLServerIntegratedSecuritySetting class represents an association between a SQL Server installation and its security settings.
MSSQL_SQLServerLanguageSetting	Association	The MSSQL_SQLServerLanguageSetting class represents an association between a SQL Server installation and its language settings.
MSSQL_SQLServerLogin	Association	The MSSQL_SQLServerLogin class represents an association between a SQL server and a login defined within the SQL Server.
MSSQL_SQLServerRegistry	Association	The MSSQL_SQLServerRegistry class represents an association between a SQL Server installation and its registry setting.
MSSQL_SQLServerServerRole	Association	The MSSQL_SQLServerServerRole class represents an association between a SQL server and server roles defined within the SQL server.
MSSQL_SQLServerSQLServerConnectionSetting	Association	The MSSQL_SQLServerSQLServerConnectionSetting class represents an association between a SQL Server installation and the settings used by the SQL Server provider to connect to the SQL Server.
MSSQL_SQLServerUser	Association	The MSSQL_SQLServerUser class represents an association between a SQL Server and a database user. This association allows an application to perform a single traversal to find the database users in a SQL Server and the login that they are mapped to.
MSSQL_StoredProcedureStoredProcedureParameter	Association	The MSSQL_StoredProcedureStoredProcedureParameter class associates a stored procedure to a parameter used in the stored procedure.
MSSQL_TableCheck	Association	The MSSQL_TableCheck class represents an association between a table and the checks defined for the table.
MSSQL_TableColumn	Association	The MSSQL_TableColumn class represents an association between a table and a column contained in the table.
MSSQL_TableFileGroup	Association	The MSSQL_TableFileGroup class represents an association between a table and the file groups used to store the table.
MSSQL_TableIndex	Association	The MSSQL_TableIndex class represents an association between a table and an index defined for the table.
MSSQL_TableKey	Association	The MSSQL_TableKey class represents an association between a table and a key defined for the table.
MSSQL_TableTextFileGroup	Association	This class associates a table with the file group that is used to store thevariable-length data in the table.
MSSQL_TableTrigger	Association	The MSSQL_TableTrigger class represents an association between a table and a trigger defined for the table.
MSSQL_TransactionLogDataFile	Association	The MSSQL_TransactionLogDataFile class represents an association between SQL Server transaction log and the Operating System file that is used to store the log.
MSSQL_UserDatabasePermission	Association	The MSSQL_UserDatabasePermission class represents the permissions granted to a user for a database. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a user has permissions to access a database by virtue of being a member of a certain database role, then there will not be a permission association instance between the user and the database.
MSSQL_UserDatatypeDefault	Association	The MSSQL_UserDatatypeDefault class represents an association between a user-defined datatype and the rule bound to the column.
MSSQL_UserDatatypeRule	Association	The MSSQL_UserDatatypeRule class represents an association between a user-defined datatype and the rule bound to the column.
MSSQL_UserLogin	Association	The MSSQL_UserLogin class represents an association between a database user and the login used to authenticate the user.
MSSQL_UserStoredProcedurePermission	Association	The MSSQL_UserStoredProcedurePermission class represents the permissions granted to a user for a stored procedure. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a user has permissions to access a stored procedure by virtue of being a member of a certain database role, then there will not be a permission association instance between the user and the stored procedure.
MSSQL_UserTablePermission	Association	The MSSQL_UserTablePermission class represents the permissions granted to a user for a table. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a user has permissions to access a table by virtue of being a member of a certain database role, then there will not be a permission association instance between the user and the table.
MSSQL_UserUserDefinedFunctionPermission	Association	The MSSQL_UserUserDefinedFunctionPermission class represents the permissions granted to a user for a stored procedure. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a user has permissions to access a user defined function by virtue of being a member of a certain database role, there will not be a permission association instance between the user and the user defined function.
MSSQL_UserViewPermission	Association	The MSSQL_UserViewPermission class represents the permissions granted to a user for a view. The instances of this class represent only the permission that has been explicitly granted or denied to the user object. For example, if a user has permissions to access a view by virtue of being a member of a certain database role, then there will not be a permission association instance between the user and the view.

Table 5.28 *The WMI SQL Classes (continued)*

Name	Type	Description
View Classes		
CIM_DataFile	Dynamic	CIM_DataFile is a type of logical file that is a named collection of data or executable code.
CIM_LogicalFile	Dynamic	The CIM_LogicalFile class represents a named collection of data (this can be executable code) located in a file system on a storage extent.
CIM_Service	Dynamic	A logical element that contains the information necessary to represent and manage the functionality provided by a device and/or software feature. A service is a general-purpose object to configure and manage the implementation of functionality. It is not the functionality itself.
Win32_BaseService	Dynamic	The Win32_BaseService class represents executable objects that are installed in a registry database maintained by the Service Control Manager. The executable file associated with a service can be started at boot time by a boot program or by the system. It can also be started on-demand by the Service Control Manager. Any service or process that is not owned by a specific user, and that provides an interface to some functionality supported by the computer system, is a descendent (or member) of this class. Example: The dynamic host configuration protocol (DHCP) client service on a Windows NT/Windows 2000 computer system.
Win32_Group	Dynamic	The Win32_Group class represents data about a group account. A group account allows access privileges to be changed for a list of users. Example: Marketing2.
Win32_Process	Dynamic	The Win32_Process class represents a sequence of events on a Win32 system. Any sequence consisting of the interaction of one or more processors or interpreters, some executable code, and a set of inputs, is a descendent (or member) of this class. Example: A client application running on a Win32 system.
Win32_Service	Dynamic	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.
Win32_UserAccount	Dynamic	The Win32_UserAccount class contains information about a user account on a Win32 system.
Win32_GroupUser	Association	The Win32_GroupUser class represents an association between a group and an account that is a member of that group.

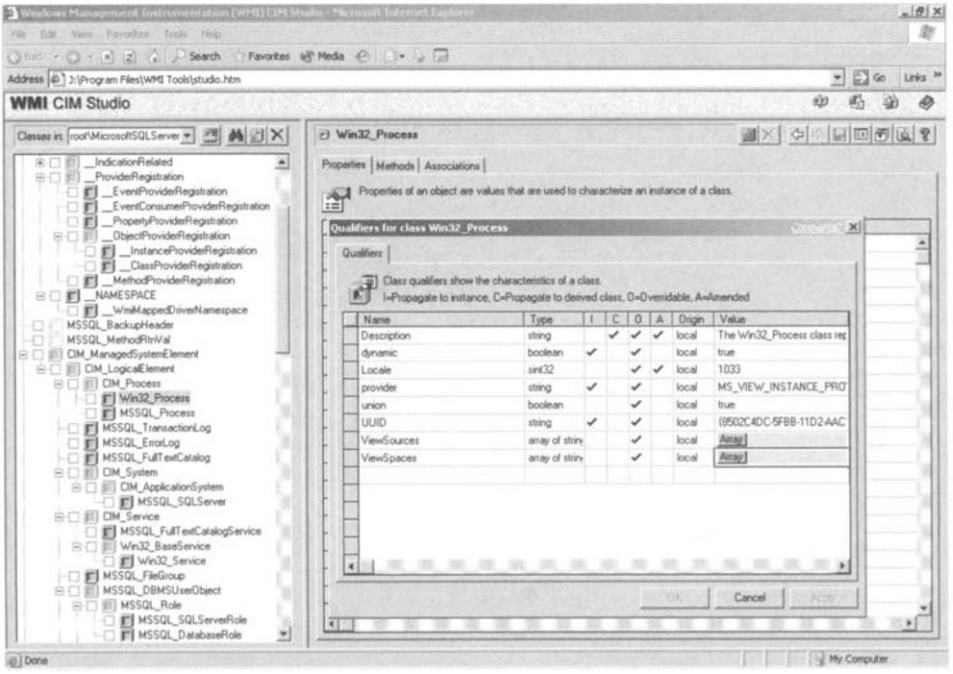

Figure 5.28 *The Win32_Process View class in the Root\MicrosoftSQLServer namespace.*

from the **Root\CIMv2** namespace available in the **Root\Microsoft-SQLServer** (see bottom of Table 5.28). Figure 5.28 shows the view class Qualifiers for the *Win32_Process* class.

5.4 WMI and some Windows applications

5.4.1 Microsoft Office

Both Office 2000 and Office XP come with their WMI instance providers. Office 2000 registers a provider called **OffProv** in the **Root\MSAPPS** namespace, while Office XP registers a provider called **OffProv10** in the **Root\MSAPPS10** namespace (see Table 5.29).

Both providers support a set of classes representing management information about Office. Office XP comes with some additional classes compared with Office 2000. Table 5.30 shows the classes available under Office 2000 and Office XP.

As shown in Table 5.29, the information about Office instances can't be updated. For instance, if you need to determine if a Word document is open, you can request the instances of the *Win32_WordDocument* class. The output would be as follows:

```
1:  C:\>GetCollectionOfInstances.wsf Win32_WordDocument /Namespace:Root\MSAPPS
2:  Microsoft (R) Windows Script Host Version 5.6
3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
4:
5:
6:  CreateDate: ............................ 20010118112800.******+***
7:  *Name: ................................. MyWordDocument.doc
8:  Path: .................................. C:\My Documents\MyWordDocument.doc
9:  Size: .................................. 295
```

Table 5.29 *The Office Provider Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete
Office 2000 and XP Providers											
OffProv (Office 2000)	Root/MSAPPS		X					X		X	
OffProv10 (Office XP)	Root/MSAPPS10		X					X		X	

Table 5.30 *The Office 2000 and Office XP Classes*

Office 2000	Office XP
N/A	Win32_Access10AlternateStartupFileLoc
N/A	Win32_Access10ComAddin
N/A	Win32_Access10ComAddins
N/A	Win32_Access10DefaultFileLoc
N/A	Win32_Access10JetComponents
N/A	Win32_Access10StartupFolder
Win32_AccessDatabase	Win32_AccessDatabase
Win32_AccessProject	Win32_AccessProject
Win32_AccessSummary	Win32_AccessSummary
N/A	Win32_ADOCoreComponents
N/A	Win32_Excel10AlternateStartupFileLoc
N/A	Win32_Excel10DefaultFileLoc
N/A	Win32_Excel10StartupFolder
Win32_ExcelActiveWorkbook	Win32_ExcelActiveWorkbook
Win32_ExcelActiveWorkbookNotable	Win32_ExcelActiveWorkbookNotable
Win32_ExcelAddIn	Win32_ExcelAddIn
Win32_ExcelAddIns	Win32_ExcelAddIns
Win32_ExcelChart	Win32_ExcelChart
Win32_ExcelCharts	Win32_ExcelCharts
N/A	Win32_ExcelComAddin
N/A	Win32_ExcelComAddins
Win32_ExcelSheet	Win32_ExcelSheet
Win32_ExcelSummary	Win32_ExcelSummary
Win32_ExcelWorkbook	Win32_ExcelWorkbook
Win32_FrontPageActivePage	Win32_FrontPageActivePage
Win32_FrontPageActiveWeb	Win32_FrontPageActiveWeb
Win32_FrontPageAddIn	Win32_FrontPageAddIn
Win32_FrontPageAddIns	Win32_FrontPageAddIns
Win32_FrontPagePageProperty	Win32_FrontPagePageProperty
Win32_FrontPageSummary	Win32_FrontPageSummary
Win32_FrontPageTheme	Win32_FrontPageTheme
Win32_FrontPageThemes	Win32_FrontPageThemes
Win32_FrontPageWebProperty	Win32_FrontPageWebProperty
N/A	Win32_JetCoreComponents
Win32_OdbcCoreComponent	Win32_OdbcCoreComponent
Win32_OdbcDriver	Win32_OdbcDriver
N/A	Win32_OfficeWatsonLog
Win32_OleDbProvider	Win32_OleDbProvider
N/A	Win32_OutlookAlternateStartupFile
N/A	Win32_OutlookComAddin
N/A	Win32_OutlookComAddins
N/A	Win32_OutlookDefaultFileLocation
N/A	Win32_OutlookStartupFolder
Win32_OutlookSummary	Win32_OutlookSummary
N/A	Win32_PowerPoint10AlternateStartupLoc
N/A	Win32_PowerPoint10ComAddin
N/A	Win32_PowerPoint10ComAddins
N/A	Win32_PowerPoint10DefaultFileLoc
N/A	Win32_PowerPoint10Font
N/A	Win32_PowerPoint10Fonts
N/A	Win32_PowerPoint10Hyperlink
N/A	Win32_PowerPoint10Hyperlinks
N/A	Win32_PowerPoint10PageNumber
N/A	Win32_PowerPoint10PageSetup
N/A	Win32_PowerPoint10SelectedTable
N/A	Win32_PowerPoint10StartupFolder
N/A	Win32_PowerPoint10Table
N/A	Win32_PowerPoint10Tables
Win32_PowerPointActivePresentation	Win32_PowerPointActivePresentation
Win32_PowerPointPresentation	Win32_PowerPointPresentation
Win32_PowerPointSummary	Win32_PowerPointSummary
N/A	Win32_Publisher10ActiveDocument
N/A	Win32_Publisher10ActiveDocumentNoTable
N/A	Win32_Publisher10AlternateStartupFileLocation
N/A	Win32_Publisher10CharacterStyle
N/A	Win32_Publisher10COMAddIn
N/A	Win32_Publisher10COMAddIns
N/A	Win32_Publisher10DefaultFileLocation
N/A	Win32_Publisher10Font
N/A	Win32_Publisher10Fonts
N/A	Win32_Publisher10Hyperlink
N/A	Win32_Publisher10Hyperlinks
N/A	Win32_Publisher10MailMerge

Table 5.30 *The Office 2000 and Office XP Classes (continued)*

Office 2000	Office XP
N/A	Win32_Publisher10PageNumber
N/A	Win32_Publisher10PageSetup
N/A	Win32_Publisher10ParagraphStyle
N/A	Win32_Publisher10Sections
N/A	Win32_Publisher10SelectedTable
N/A	Win32_Publisher10StartupFolder
N/A	Win32_Publisher10Styles
N/A	Win32_Publisher10Table
N/A	Win32_Publisher10Tables
Win32_PublisherSummary	Win32_PublisherSummary
N/A	Win32_RDOCoreComponents
Win32_ServerExtension	Win32_ServerExtension
Win32_Transport	Win32_Transport
Win32_WebConnectionError	Win32_WebConnectionError
Win32_WebConnectionErrorMessage	Win32_WebConnectionErrorMessage
Win32_WebConnectionErrorText	Win32_WebConnectionErrorText
Win32_WordActiveDocument	Win32_Word10ActiveDocument
Win32_WordActiveDocumentNotable	Win32_Word10ActiveDocumentNotable
Win32_WordAddin	Win32_Word10Addin
N/A	Win32_Word10AlternateStartupFileLocation
Win32_WordCharacterStyle	Win32_Word10CharacterStyle
N/A	Win32_Word10DefaultFileLocation
Win32_WordDocument	Win32_Word10Document
Win32_WordField	Win32_Word10Field
Win32_WordFields	Win32_Word10Fields
Win32_WordFileConverter	Win32_Word10FileConverter
Win32_WordFileConverters	Win32_Word10FileConverters
Win32_WordFont	Win32_Word10Font
Win32_WordFonts	Win32_Word10Fonts
Win32_WordHeaderAndFooter	Win32_Word10HeaderAndFooter
Win32_WordHyperlink	Win32_Word10Hyperlink
Win32_WordHyperlinks	Win32_Word10Hyperlinks
Win32_WordMailMerge	Win32_Word10MailMerge
Win32_WordPageNumber	Win32_Word10PageNumber
Win32_WordPageSetup	Win32_Word10PageSetup
Win32_WordParagraphStyle	Win32_Word10ParagraphStyle
Win32_WordSections	Win32_Word10Sections
Win32_WordSelectedTable	Win32_Word10SelectedTable
Win32_WordSettings	Win32_Word10Settings
N/A	Win32_Word10StartupFileLocation
Win32_WordStyles	Win32_Word10Styles
Win32_WordSummary	Win32_Word10Summary
Win32_WordTable	Win32_Word10Table
Win32_WordTables	Win32_Word10Tables
Win32_WordTemplate	Win32_Word10Template
N/A	Win32_WordComAddin
N/A	Win32_WordComAddins

5.4.2 Internet Explorer

Internet Explorer 5.0 (and later) provides a WMI instance provider called **IEINFO5**. This provider supports a collection of classes representing the configuration settings of Internet Explorer. These classes are available in the **Root\CIMV2\Applications\MicrosoftIE** namespace (see Table 5.31).

As with the *Office* provider, the *Internet Explorer* provider doesn't support update of the configuration settings. Therefore, it is a read-only mode provider (see Table 5.32).

The *MicrosoftIE_LanSettings* class exposes the same information as the *Win32_Proxy* class. However, with the *SetProxySetting* method exposed by the *Win32_Proxy* class, it is possible to update the Proxy LAN settings. You

Table 5.31 *The Internet Explorer Provider Capabilities*

Provider Name	Provider Namespace	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete	Windows Server 2003	Windows XP	Windows 2000 Server	Windows 2000 Professional	Windows NT 4.0
Internet Explorer Provider																
ieinfo5	Root/CIMv2/Applications/MicrosoftIE		X					X		X		X	X	X	X	

can refer to Chapter 3 (Sample 3.13, "Managing the Windows Proxy LAN settings") for more information about the *Win32_Proxy* class.

For instance, if you need to know the certificates available from Internet Explorer, you can request the instances of the *MicrosoftIE_Certificate* class. The output would be as follows:

```
C:\>GetCollectionOfInstances.wsf MicrosoftIE_Certificate
                                 /Namespace:Root\CIMv2\Applications\MicrosoftIE
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.


*IssuedBy: ............................... Hewlett-Packard Primary
                                           Class 2 Certification Authority
*IssuedTo: ............................... Alain Lissoir
SignatureAlgorithm: ...................... md5RSA
*Type: ................................... Personal
Validity: ................................ 21-05-2002 to 22-05-2003

*IssuedBy: ............................... Administrator
*IssuedTo: ............................... Administrator
SignatureAlgorithm: ...................... sha1RSA
*Type: ................................... Personal
Validity: ................................ 01-08-2002 to 31-07-2005

*IssuedBy: ............................... Administrator
*IssuedTo: ............................... Administrator
SignatureAlgorithm: ...................... sha1RSA
*Type: ................................... Personal
Validity: ................................ 01-08-2002 to 08-07-2102
```

Table 5.32 *The Internet Explorer WMI Classes*

Name	Description
MicrosoftIE_Summary	Retrieves the various settings of Internet Explorer, such as the active printer, the build number, Java Virtual Machine version, Service Pack levels and fixes, version, etc.
MicrosoftIE_ConnectionSettings	Retrieves the Dial-up connection settings of Internet Explorer.
MicrosoftIE_Object	Retrieves information about the various ActiveX, Codecs, Objects added to Internet Explorer.
MicrosoftIE_Publisher	Retrieves Internet Explorer publisher information.
MicrosoftIE_LanSettings	Retrieves the LAN connection settings of Internet Explorer.
MicrosoftIE_FileVersion	Retrieves Internet Explorer files information (date, version, etc.).
MicrosoftIE_Certificate	Retrieves Internet Explorer certificates information.
MicrosoftIE_ConnectionSummary	Retrieves Internet Explorer connection summary.
MicrosoftIE_Cache	Retrieves Internet Explorer cache settings.
MicrosoftIE_Security	Retrieves Internet Explorer security settings.

As another example, if you need to determine the file version used by Internet Explorer, you can request the instances of the *MicrosoftIE_FileVersion* class. The output would be as follows:

```
C:\>GetCollectionOfInstances.wsf MicrosoftIE_FileVersion
                                 /Namespace:Root\CIMv2\Applications\MicrosoftIE
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.


Company: ............................... Microsoft Corporation
Date: ................................... 20011112140000.******+***
*File: .................................. actxprxy.dll
*Path: .................................. J:\WINDOWS\system32
Size: ................................... 96
Version: ................................ 6.0.3590.0

Company: ............................... Microsoft Corporation
Date: ................................... 20011112140000.******+***
*File: .................................. advpack.dll
*Path: .................................. J:\WINDOWS\system32
Size: ................................... 89
Version: ................................ 6.0.3590.0

Company: ............................... Microsoft Corporation
Date: ................................... 20011112140000.******+***
*File: .................................. asctrls.ocx
*Path: .................................. J:\WINDOWS\system32
Size: ................................... 87.5
Version: ................................ 6.0.3590.0

Company: ............................... Microsoft Corporation
Date: ................................... 20011112140000.******+***
*File: .................................. browselc.dll
*Path: .................................. J:\WINDOWS\system32
Size: ................................... 61.5
Version: ................................ 6.0.3590.0
...:
...:
...:
```

5.5 WMI and some Enterprise Management software

Most management software available today on the market utilizes WMI information. By developing scripts on top of the WMI COM API and discovering what WMI information is available from Windows 2000, Windows Server 2003, or Windows XP, we learned how to use WMI information from scripts. The great thing about WMI is that it abstracts the management information for all WMI information types, which means that management information accessed from scripts can be accessed in the same way from any other types of information. Therefore, all WMI information

knowledge acquired throughout this book is directly applicable to any Enterprise Management software supporting WMI information utilization.

Considering Enterprise Management software as some powerful WMI consumers is just a partial view of the WMI capabilities offered by some software solutions. Actually, most Enterprise Management software also provides information through WMI, implementing WMI providers and Common Information Model (CIM) class extensions in order to provide management data. Of course, the WMI information available depends heavily on the software implementation. Today, all major management solutions for the Windows platform rely on WMI, such as Microsoft Operations Manager (http://www.microsoft.com/mom), Systems Management Server (http://www.microsoft.com/smserver/default.asp), IBM's Tivoli solution portfolio (http://www-3.ibm.com/software/tivoli/), Hewlett-Packard's OpenView (http://www.openview.hp.com), Computer Associates Unicenter (http://www.cai.com/products), and even Compaq Insight Manager (http://www.compaq.com/manage). Unfortunately, it will be impossible to review the WMI capabilities of all Enterprise Management software available on the market today. Therefore, we will only consider the following Enterprise Management software:

- Compaq Insight Manager and Management Agents (now called Insight Manager after the HP/Compaq merger)
- Microsoft Operation Manager (MOM)
- HP OpenView Operations for Windows (OVOW)

Moreover, only the WMI provider aspects of the software solution will be considered, since they are the most interesting information to know about in order to utilize the WMI information they provide. If you use another management software solution, please contact your Enterprise Management software vendor to gather more information on this subject.

5.5.1 Insight Management Agents

The Insight Management Agents were developed by Compaq to manage the Compaq server's hardware. The complete management solution, also known under the name "Compaq Insight Manager," is made up of a management console and a series of agents.

Information retrieved by the agents can be very useful, because sometimes hardware will obscure things from the Operating System. For example, Windows can see logical drives created on a SmartArray controller but has no information about the physical drives or the RAID level. The Insight

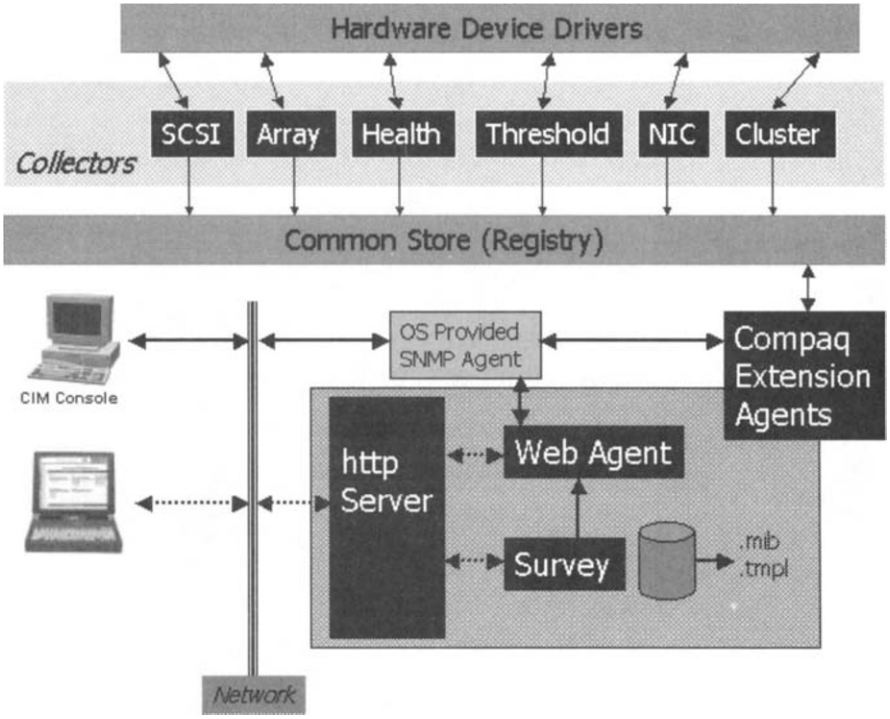

Figure 5.29 *The Insight Management Web-enabled Agents architecture of CIM, version 7 SP1.*

Management Agents provide information about this and many other aspects of a ProLiant. The management console consolidates the information collected by the Insight Management Agents. The architecture of the Insight Management Agents, coming with "Insight Manager version 7 SP1," is based on SNMP and HTTP and does not currently act as WMI providers. Although there is a WMI consumer component in the agents that does provide some limited information about Windows, no hardware-specific information is exposed through WMI with this version (see Figure 5.29). This data is routinely retrieved via SNMP.

In Chapter 3, section 3.7.5.2, we saw how to integrate and access standard SNMP information (SNMPv1 and SNMPv2) from WMI. We also examined how private MIB information can be integrated (Cisco MIB) and how SNMP traps can be received. By applying this knowledge and integrating the SNMP information of the Insight Management Agents in WMI, we can use the same techniques to retrieve this information with WMI scripts (or any other WMI consumers). As a result, the Agents SNMP information

of Insight Manager version 7 SP1 is accessed through the Windows WMI *SNMP* providers.

Because the Insight Agent SNMP information is part of a private MIB, it is obvious that we may find some specific SNMP information that is not defined in the standard RFC 1213. This Enterprise MIB information is defined in a set of MIB files available via download from the Management Toolkit or on the management CD in the ProLiant Essentials Value Pack included with every ProLiant.

To access the enterprise MIB information from WMI, it is necessary to load the MIB file information into the WMI CIM repository. This can be done using the **SMI2SMIR.Exe** MIB to MOF converter provided with WMI.

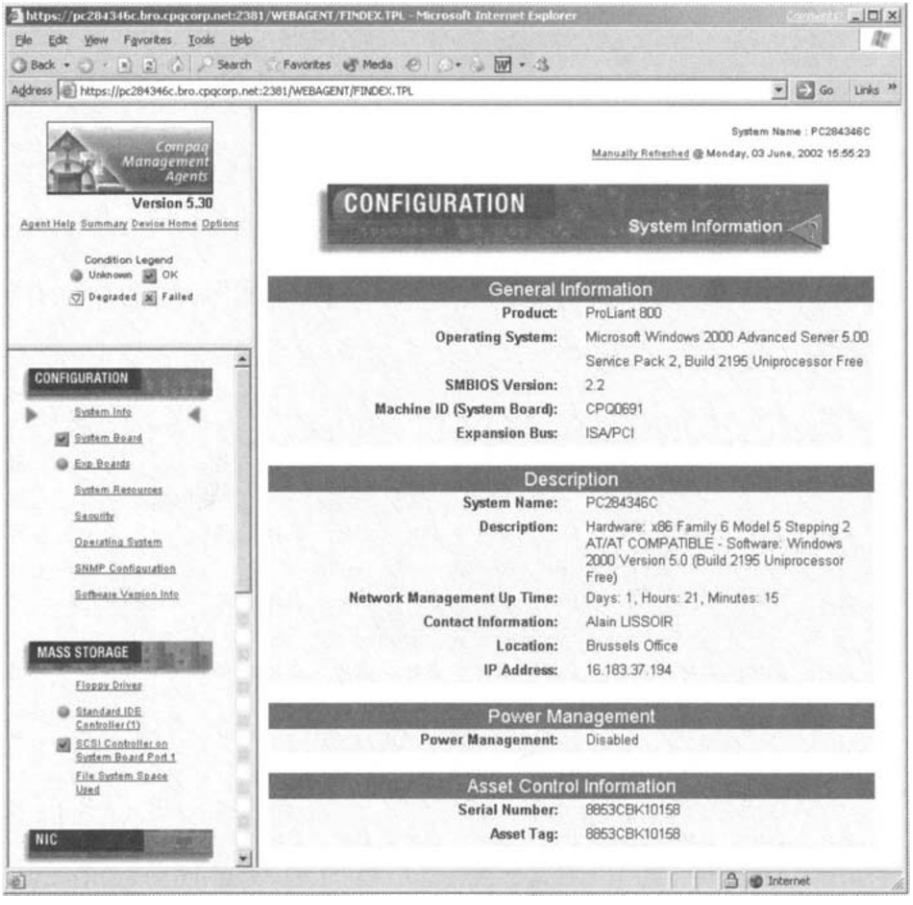

Figure 5.30 *The Insight Management Agents information.*

Table 5.33 *Insight SNMP MIB Files*

MIB Filename	Description	Compaq Insight Manager Support
CPQAPLI.MIB	Compaq Appliance MIB	v 4.70
CPQCLUS.MIB	Compaq Cluster MIB	v 4.21
CPQCMC.MIB	Compaq/Powerware/Rittal MIB	v 5.30
CPQCR.MIB	ClusteredRaid MIB (Shrewsbury)	v 4.50
CPQFCA.MIB	Compaq Fibre Channel MIB	v 3.60
CPQHLTH.MIB	Compaq System Health MIB	v 2.x
CPQHOST.MIB	Compaq Host MIB	v 2.x
CPQIDA.MIB	Compaq Drive Array MIB	v 2.x
CPQIDE.MIB	Compaq IDE MIB	v 2.51
CPQNIC.MIB	Compaq NIC MIB	v 4.20
CPQRACK.MIB	Compaq Rack MIB	v 5.30
CPQRECOV.MIB	Compaq Recovery Server MIB	v 2.50
CPQSCSI.MIB	Compaq SCSI MIB	v 2.x
CPQSINFO.MIB	Compaq System Information MIB	v 2.x
CPQSM2.MIB	Compaq Remote Insight MIB	v 2.51
CPQSTAT.MIB	Compaq External Status MIB	v 3.50
CPQSTDEQ.MIB	Compaq Standard Equipment MIB	v 2.x
CPQSTSYS.MIB	Compaq Storage System MIB	v 2.x
CPQTHRSH.MIB	Compaq Threshold MIB	v 2.x
CPQUPS.MIB	Compaq UPS MIB	v 2.x
SVRCLU.MIB	Compaq Common Cluster Management MIB	v 4.70
SVRNTC.MIB	Digital Microsoft Windows Cluster MIB	v 4.70

The Insight Agents have a lot of enterprise MIB information, so let's take a very easy example. Figure 5.30 shows some information that is specific to ProLiant hardware. The figure shows the system type with its serial number and its asset tag.

To retrieve this information from WMI via SNMP, we must a use a private MIB file. The required MIB file is called "CPQSINFO.MIB." You can check Table 5.33 for a complete list of the SNMP MIB files. To successfully convert the "CPQSINFO.MIB" MIB file into a MOF file, we also need the "CPQHOST.MIB" and "RFC1213.MIB" MIB files, since they contain definitions used in "CPQSINFO.MIB." The conversion to a MOF file can be created as follows:

```
C:\>SMI2SMIR /m 0 /g CPQSINFO.MIB CPQHOST.MIB Rfc1213.MIB > CPQSINFO.MOF

SMI2SMIR : Version <UnknownVersion> : MIB definitions compiled from "CPQSINFO.MIB"

SMI2SMIR : Syntax Check successful on "CPQSINFO.MIB"

SMI2SMIR : Version <UnknownVersion> : MIB definitions compiled from "CPQHOST.MIB"

SMI2SMIR : Syntax Check successful on "CPQHOST.MIB"

SMI2SMIR : Version <UnknownVersion> : MIB definitions compiled from "Rfc1213.MIB"

SMI2SMIR : Syntax Check successful on "Rfc1213.MIB"

SMI2SMIR : Semantic Check successful on "CPQSINFO.MIB"

SMI2SMIR: Generated MOF successfully
```

Once the MOF file is generated, it is ready to be loaded in the WMI CIM repository with **MOFComp.Exe**, as follows:

```
C:\>Mofcomp.Exe CPQSINFO.MOF
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: CPQSINFO.MOF
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

Once completed, the selected namespace to access the Insight Management Agents should show the WMI classes representing the Insight Management Agents information. Ensure that the Insight Management Agents are properly installed and that the SNMP host is available on the network during this operation (see Figure 5.31).

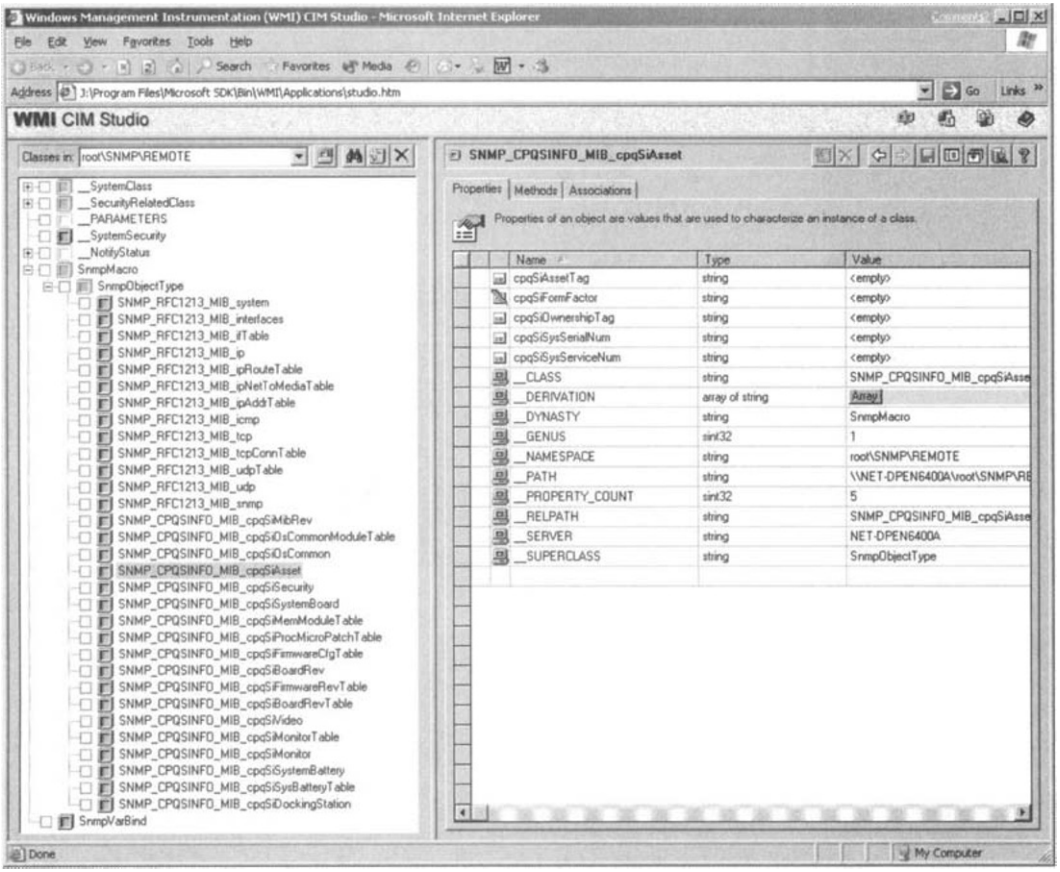

Figure 5.31 *Some Insight Management Agents SNMP classes in the CIM repository.*

If we reuse a sample script developed in Chapter 1 (Sample 1.4, "Listing a single instance of a class with its properties formatted") to retrieve the instances of the *SNMP_CPQSINFO_MIB_cpqSiAsset* class, we should obtain an output similar to this one:

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


- SNMP_CPQSINFO_MIB_cpqSiAsset ------------------------------------------------
cpqSiAssetTag: ......................... 8853CBK10158
cpqSiFormFactor: ....................... tower
cpqSiOwnershipTag: ..................... 2020202020202020202020202020202020...
cpqSiSysSerialNum: ..................... 8853CBK10158
```

We recognize the serial number (line 10) and the asset tag (line 7) from Figure 5.30. Of course, we only compiled one single MOF file. To create all MOF files representing the complete Insight Manager v7 SP1 private MIB, you can use the following batch file:

```
SMI2SMIR /m 0 /g MIBS\CPQSINFO.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSINFO.MOF
SMI2SMIR /m 0 /g MIBS\CPQAPLI.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQAPLI.MOF
SMI2SMIR /m 0 /g MIBS\CPQCLUS.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQCLUS.MOF
SMI2SMIR /m 0 /g MIBS\CPQCMC.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQCMC.MOF
SMI2SMIR /m 0 /g MIBS\CPQCR.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQCR.MOF
SMI2SMIR /m 0 /g MIBS\CPQFCA.MIB MIBS\CPQSTSYS.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB
                 > CPQFCA.MOF
SMI2SMIR /m 0 /g MIBS\CPQHLTH.MIB MIBS\CPQSINFO.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB
                 > CPQHLTH.MOF
SMI2SMIR /m 0 /g MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQHOST.MOF
SMI2SMIR /m 0 /g MIBS\CPQIDA.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQIDA.MOF
SMI2SMIR /m 0 /g MIBS\CPQIDE.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQIDE.MOF
SMI2SMIR /m 0 /g MIBS\CPQNIC.MIB MIBS\CPQSTDEQ.MIB MIBS\CPQSINFO.MIB MIBS\CPQHOST.MIB
                 MIBS\RFC1213.MIB > CPQNIC.MOF
SMI2SMIR /m 0 /g MIBS\CPQRACK.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQRACK.MOF
SMI2SMIR /m 0 /g MIBS\CPQRECOV.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQRECOV.MOF
SMI2SMIR /m 0 /g MIBS\CPQSCSI.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSCSI.MOF
SMI2SMIR /m 0 /g MIBS\CPQSINFO.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSINFO.MOF
SMI2SMIR /m 0 /g MIBS\CPQSM2.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSM2.MOF
SMI2SMIR /m 0 /g MIBS\CPQSTAT.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSTAT.MOF
SMI2SMIR /m 0 /g MIBS\CPQSTDEQ.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSTDEQ.MOF
SMI2SMIR /m 0 /g MIBS\CPQSTSYS.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQSTSYS.MOF
SMI2SMIR /m 0 /g MIBS\CPQTHRSH.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQTHRSH.MOF
SMI2SMIR /m 0 /g MIBS\CPQUPS.MIB MIBS\CPQHOST.MIB MIBS\RFC1213.MIB > CPQUPS.MOF
SMI2SMIR /m 0 /g MIBS\ETHER.MIB > ETHER.MOF
SMI2SMIR /m 0 /g MIBS\SVRCLU.MIB > SVRCLU.MOF
SMI2SMIR /m 0 /g MIBS\SVRNTC.MIB > SVRNTC.MOF
SMI2SMIR /m 0 /g MIBS\TOKEN.MIB > TOKEN.MOF
```

Once all generated MOF files are loaded in the CIM repository, only the MIBs that the correlation mechanism can resolve will be visible in the CIM repository. You can get more information about the created classes from Table 5.34.

Table 5.34 *Insight Management Agents WMI Classes Created from the MIB Files*

WMI Class name	Description
SNMP_CPQSTDEQ_MIB_cpqSeMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQSTDEQ_MIB_cpqSeOsCommon	The Insight Agent's polling frequency. The frequency, in seconds, at which the Insight Agent requests information from the device driver. A frequency of zero (0) indicates that the Insight Agent retrieves the information upon request of a management station; it does not poll the device driver at a specific interval.
SNMP_CPQSTDEQ_MIB_cpqSeOsCommonModuleTable	A table of software modules that provide an interface to the device this MIB describes. A description of a software module that provides an interface to the device this MIB describes.
SNMP_CPQSTDEQ_MIB_cpqSeCpuTable	A list of the CPUs (processors) in the system. The main processor (if such a concept is valid for this machine) should be the first entry in the table. A description of a CPU (processor) in the system.
SNMP_CPQSTDEQ_MIB_cpqSeCpuCacheTable	A list of the CPU caches in the system. A description of a CPU caches in the system.
SNMP_CPQSTDEQ_MIB_cpqSeFpuTable	A list of the FPUs (floating-point coprocessors) in the system. A description of an FPU in the system. The fpuUnitIndex of any entry in this table will equal the cpuUnitIndex of the corresponding CPU in the cpqSeCpu table.
SNMP_CPQSTDEQ_MIB_cpqSeMemory	The amount of base memory and the total amount of memory in kilobytes. A kilobyte is 1024 bytes.
SNMP_CPQSTDEQ_MIB_cpqSeEisaInitTable	A list of EISA function port initialization entries. A description of an EISA function port initialization.
SNMP_CPQSTDEQ_MIB_cpqSeEisaSlotTable	A list of EISA slot information entries. A description of an EISA slot.
SNMP_CPQSTDEQ_MIB_cpqSeEisaIntTable	A list of EISA function interrupt configuration entries. A description of an EISA function interrupt configuration.
SNMP_CPQSTDEQ_MIB_cpqSeEisaPortTable	A list of EISA function port I/O configuration entries. A description of an EISA function port I/O configuration.
SNMP_CPQSTDEQ_MIB_cpqSeEisaFreeFormTable	A list of EISA function free form entries. A description of an EISA function free form.
SNMP_CPQSTDEQ_MIB_cpqSeEisaDmaTable	A list of EISA function DMA configuration entries. A description of an EISA function DMA configuration.
SNMP_CPQSTDEQ_MIB_cpqSeEisaFunctTable	A list EISA function information entries. A description of an EISA function.
SNMP_CPQSTDEQ_MIB_cpqSeEisaMemTable	A list of EISA function memory configuration entries. A description of an EISA function memory configuration.
SNMP_CPQSTDEQ_MIB_cpqSeOptRomTable	A table of option ROM descriptions. An option ROM description.
SNMP_CPQSTDEQ_MIB_cpqSeRom	The BIOS ROM data area, System ROM version information for the redundant ROM image (this will be left blank if the system does not support a redundant ROM) and the system ROM version information.
SNMP_CPQSTDEQ_MIB_cpqSeKeyboard	A description of the keyboard.
SNMP_CPQSTDEQ_MIB_cpqSeVideo	A description of the video system in the computer. This may include the manufacturer, board name, modes supported, etc..
SNMP_CPQSTDEQ_MIB_cpqSeSerialPortTable	A table of serial port descriptions. A description of a serial port.
SNMP_CPQSTDEQ_MIB_cpqSeParallelPortTable	A table of parallel port descriptions. A description of a parallel port.
SNMP_CPQSTDEQ_MIB_cpqSeFloppyDiskTable	A table of floppy drive descriptions. A floppy drive description.
SNMP_CPQSTDEQ_MIB_cpqSeFixedDiskTable	A table of ST-506 interface accessible fixed disk descriptions. A fixed disk description.
SNMP_CPQSTDEQ_MIB_cpqSePciFunctTable	A list of PCI function information entries. A description of the functions in each PCI slot.
SNMP_CPQSTDEQ_MIB_cpqSePciSlotTable	A list of PCI slot information entries. A description of a PCI slot.
SNMP_CPQSTDEQ_MIB_cpqSePciMemoryTable	A list of PCI base memory entries. A description of the base memory usage in each function.
SNMP_CPQSINFO_MIB_cpqSiMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQSINFO_MIB_cpqSiOsCommonModuleTable	A table of software modules that provide an interface to the device this MIB describes. A description of a software module that provides an interface to the device this MIB describes.
SNMP_CPQSINFO_MIB_cpqSiOsCommon	The Insight Agent's polling frequency. The frequency, in seconds, at which the Insight Agent requests information from the device driver. A frequency of zero (0) indicates that the Insight Agent retrieves the information upon request of a management station, it does not poll the device driver at a specific interval.
SNMP_CPQSINFO_MIB_cpqSiAsset	Contains miscellaneous information about the system, such as the customer changeable identifier (asset) that is set to the system serial number at the time of manufacture, the form factor of the system, the serial number of the system unit, and the service number of the system unit.
SNMP_CPQSINFO_MIB_cpqSiSecurity	Contains miscellaneous information about the system security, such as the state of the diskette boot control feature or the current configuration of the Compaq Smart Cover sensor.
SNMP_CPQSINFO_MIB_cpqSiSystemBoard	Contains miscellaneous information about the system board, such as the state of the auxiliary input (pointing) device or the current condition of the correctable memory.
SNMP_CPQSINFO_MIB_cpqSiMemModuleTable	A table of memory module descriptions. A memory module description.
SNMP_CPQSINFO_MIB_cpqSiProcMicroPatchTable	This table lists the set of processor microcode patches that the system ROM contains. During post, the ROM will apply the appropriate patches to the CPU microcode. Scan this table to ensure that a patch is being applied to a processor. An entry describing one microcode patch contained in the system ROM.
SNMP_CPQSINFO_MIB_cpqSiFirmwareCfgTable	Table of soft switches and symbols maintained by the firmware. May be operating system and/or option-specific and will certainly be system-specific. This list is intended to be easily extensible and support arbitrary datatypes. It includes such switches as powerup options, default dump device, etc. Note: The string comparision for svrFwSymbolName is case insensitive. Each entry represents one variable or symbol maintained by or stored by some instance of firmware in the system.
SNMP_CPQSINFO_MIB_cpqSiBoardRev	The previous and current board revision configuration date in MM/DD/YY format. This is the date that the EISA Configuration Utility was used to define the current configuration.
SNMP_CPQSINFO_MIB_cpqSiFirmwareRevTable	A table of firmware revision descriptions. A firmware revision description.
SNMP_CPQSINFO_MIB_cpqSiBoardRevTable	A table of board revision descriptions. A board revision description.
SNMP_CPQSINFO_MIB_cpqSiVideo	The manufacturer, model description and technical information of the video display.
SNMP_CPQSINFO_MIB_cpqSiMonitorTable	A table of all video monitor descriptions connected to this system. A video monitor description.
SNMP_CPQSINFO_MIB_cpqSiMonitor	This value specifies the overall condition of the system's monitor(s).

Table 5.34 *Insight Management Agents WMI Classes Created from the MIB Files (continued)*

WMI Class name	Description
SNMP_CPQSINFO_MIB_cpqSiSystemBattery	This value specifies the overall condition of all of the system batteries.
SNMP_CPQSINFO_MIB_cpqSiSysBatteryTable	A table of System Batteries. A system battery description.
SNMP_CPQSINFO_MIB_cpqSiDockingStation	Docking station miscellaneous information such as the asset tag, the serial number, status, and model.
SNMP_CPQSCSI_MIB_cpqScsiMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQSCSI_MIB_cpqScsiOsCommonModuleTable	A table of software modules that provide an interface to the device this MIB describes. A description of software modules that provide an interface to the device this MIB describes.
SNMP_CPQSCSI_MIB_cpqScsiOsCommon	The Insight Agent's polling frequency. The frequency, in seconds, at which the Insight Agent requests information from the device driver. A frequency of zero indicates that the Insight Agent retrieves the information upon request of a management station; it does not poll the device driver at a specific interval.
SNMP_CPQSCSI_MIB_cpqScsiCntlrTable	Compaq SCSI Controller Table. Compaq SCSI Controller Table Entry.
SNMP_CPQSCSI_MIB_cpqScsiPhyDrvTable	Compaq SCSI Physical Drive Table. Compaq SCSI Physical Drive Entry.
SNMP_CPQSCSI_MIB_cpqScsiTargetTable	Compaq SCSI Target Table. Compaq SCSI Target Entry.
SNMP_CPQHLTH_MIB_cpqHeMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQHLTH_MIB_cpqHeThermalFanTable	A table of fan descriptions. A fan description.
SNMP_CPQHLTH_MIB_cpqHeFltTolFanTable	A table of Fault-Tolerant Fan Entries. A Fault-Tolerant Fan Entry.
SNMP_CPQHLTH_MIB_cpqHeTemperatureTable	A table of Temperature Sensor Entries. A Temperature Sensor Entry.
SNMP_CPQHLTH_MIB_cpqHeThermal	Miscellaneous information about the thermal envrionment, such as the overall condition of the system's thermal environment or the status of the fan(s) in the system.
SNMP_CPQHLTH_MIB_cpqHeSysUtilPciTable	A table of PCI utilization numbers for a whole aggregate PCI bus or a specific device on that bus. PCI utilization entry.
SNMP_CPQHLTH_MIB_cpqHeSysUtil	The EISA bus utilization as a percentage of the theoretical maximum during the last hour, the last 30 minutes, the last 5 minutes, and the last minute. The total time (in minutes) the system has been in full operation (while the server health supporting software was running) is also available.
SNMP_CPQHOST_MIB_cpqHoMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQHOST_MIB_cpqHoInfo	A further description of the host OS.
SNMP_CPQHOST_MIB_cpqHoCpuUtilTable	A table of CPU utilization entries. A description of a CPU's utilization.
SNMP_CPQHOST_MIB_cpqHoFileSysTable	A table of file system descriptions. A file system description.
SNMP_CPQHOST_MIB_cpqHoIfPhysMap	The overall condition of all interfaces.
SNMP_CPQHOST_MIB_cpqHoIfPhysMapTable	A table of interface to physical hardware mappings. A mapping of an interface table entry to physical hardware.
SNMP_CPQHOST_MIB_cpqHoSystemStatus	This item indicates how many code server shares are currently configured on the system. The date/time when the agents were last loaded. The globally unique identifier of this server. If the OS cannot determine a unique ID, it will default the variable to contain all 0's. The management station can then perform a SET to this variable to provide the unique ID.
SNMP_CPQIDE_MIB_cpqIdeMibRev	The major and minor revision level of the MIB with the overall condition (1:other, 2:ok, 3:degraded, 4:failed).
SNMP_CPQIDE_MIB_cpqIdeOsCommonModuleTable	A table of software modules that provide an interface to the device this MIB describes. A description of software modules that provide an interface to the device this MIB describes.
SNMP_CPQIDE_MIB_cpqIdeOsCommon	The Insight Agent's polling frequency. The frequency, in seconds, at which the Insight Agent requests information from the device driver. A frequency of zero indicates that the Insight Agent retrieves the information upon request of a management station; it does not poll the device driver at a specific interval.
SNMP_CPQIDE_MIB_cpqIdeIdentTable	Compaq IDE Drive Identification Table. Compaq IDE Identification Table Entry.

This technique is applicable to any Insight Management Agents version. However, the Insight Management Agents also expose some performance data through WMI in the **Root\Default** namespace. Table 5.35 shows the list of classes and their related properties with the Agents version exposing that information. The ones marked with "< 5.5" were first exposed via WMI in version 5.5 but were also previously exposed through the BMC Legacy Patrol software.

All classes are derived from the *CPQ_System_Performance* superclass (see Figure 5.32).

Table 5.35 *The Insight Management Agents Performance Classes*

System Object	CPQ_System	Counter available from Agent version
System performance properties supported are:		
System Up Time	SystemUpTime	< 5.5
Total Threads	TotalThreads	< 5.5
Context Switch Rate	ContextSwitchRate	< 5.5
Processor Queue Length	CPUQueueLength	< 5.5
Total Processes	Processes	5.5
Registry In Use Percent	RegistryUsage	5.5
Server Object	**CPQ_Server**	
Server performance properties supported are:		
Network Utilization	TotalByteRate	< 5.5
Server Sessions	ServerSessions	< 5.5
Access Permission Errors	AccessPermissionsErrors	< 5.5
Access Granted Errors	GrantedAccessErrors	< 5.5
Logon Errors	LogonErrors	< 5.5
Sessions Errored-Out	SessionsErroredOut	5.5
Context Block Queue Rate	ContextBlockQueueRate	5.5
Processor Object	**CPQ_Processor**	
Processor performance properties supported are:		
Interrupts Per Second	InterruptRate	< 5.5
Processor Time Percent	CpuTimePercent	< 5.5
Processor User Time Percent	CpuUserTimePercent	< 5.5
Processor Privileged Time Percent	PrivilegedCpuTimePercent	< 5.5
Percent DPC Time	PercentDPCTime	6.2
Percent Interrupt Time	PercentInterruptTime	6.2
Memory Object	**CPQ_Memory**	
Memory performance properties supported are:		
Available Memory	AvailableBytes	< 5.5
Pages Rate	PageRate	< 5.5
Pages Input Rate	PageInputRate	< 5.5
Pages Output Rate	PageOutputRate	5.5
Page Fault Rate	PageFaultRate	< 5.5
Cache Faults Rate	CacheFaultRate	< 5.5
Page Reads/Sec	PageReadsPersec	6.2
Page Writes/Sec	PageWritesPersec	6.2
Pool Nonpaged Bytes	PoolNonpagedBytes	6.2
Cache Bytes	CacheBytes	6.2
Paging File Object	**CPQ_PagingFile**	
Paging File performance properties supported are:		
Paging File Instance Name	PagingFile	< 5.5
Paging File Usage Percent	PageFileUsagePercent	< 5.5
Cache Object	**CPQ_Cache**	
Cache performance properties supported are:		
Copy Read Hits Percent	CopyReadHitsPercent	< 5.5
Cache Copy Reads Rate	CopyReadRate	< 5.5
Physical Disk Object		
For each physical disk instance:		
Disk Instance Name	PhysicalDisk	< 5.5
Average Disk Queue Length	DiskQueueLength	< 5.5
Disk Busy Time Percent	DiskTimePercent	< 5.5
Disk Bytes/Sec	DiskBytesPersec	6.2
Disk Transfers/Sec	DiskTransfersPersec	6.2
Disk Reads/Sec	DiskReadsPersec	6.2
Disk Writes/Sec	DiskWritesPersec	6.2
Disk Read Bytes/Sec	DiskReadBytesPersec	6.2
Disk Write Bytes/Sec	DiskWriteBytesPersec	6.2
Current Disk Queue Length	CurrentDiskQueueLength	6.2

Table 5.35 *The Insight Management Agents Performance Classes (continued)*

Logical Disk Object	**CPQ_LogicalDisk**	
Logical Disk performance properties supported are:		
Disk Instance Name	LogicalDisk	< 5.5
Disk Free Space	FreeMegabytes	< 5.5
Disk Free Space Percent	FreeSpacePercent	< 5.5
Average Disk Queue Length	DiskQueueLength	< 5.5
Disk Busy Time Percent	DiskTimePercent	< 5.5
Network Interface Object	**CPQ_NetworkInterface**	
Network Interface performance properties supported are:		
Controller Name	NetworkInterface	< 5.5
Total Byte Rate	TotalByteRate	< 5.5
Packet Rate	PacketRate	< 5.5
Output Queue Length	OutputQueueLength	< 5.5
Packet Outbound Errors	PacketOutboundErrs	5.5
Packet Receive Errors	PacketReceiveErrs	5.5
Current Bandwidth	CurrentBandwidth	5.5
Bytes Sent/Sec	BytesSentPersec	6.2
Bytes Received/Sec	BytesReceivedPersec	6.2
Packets Sent/Sec	PacketsSentPersec	6.2
Packets Received/Sec	PacketsReceivedPersec	6.2
TCP Object	**CPQ_Tcp**	
TCP performance properties supported are:		
Active Connections	ConnectionsActive	< 5.5
Established Connections	ConnectionsEstablished	< 5.5
Segments Rate	SegmentsRate	< 5.5
Retransmitted Segment Rate	SegmentsRetransmitRate	< 5.5
Connection Failures	ConnectionFailures	< 5.5
Process Object	**CPQ_Process**	
Process performance properties supported are:		
Process Name	Process	< 5.5
Thread Count	ThreadCount	< 5.5
Private Bytes	PrivateBytes	< 5.5
Page File Bytes	PageFileBytes	< 5.5
Working Set	WorkingSet	< 5.5
Processor Time Percent	CpuTimePercent	< 5.5
Processor Privileged Time Percent	PrivilegedTimePercent	< 5.5
Page Fault Rate	PageFaultRate	< 5.5

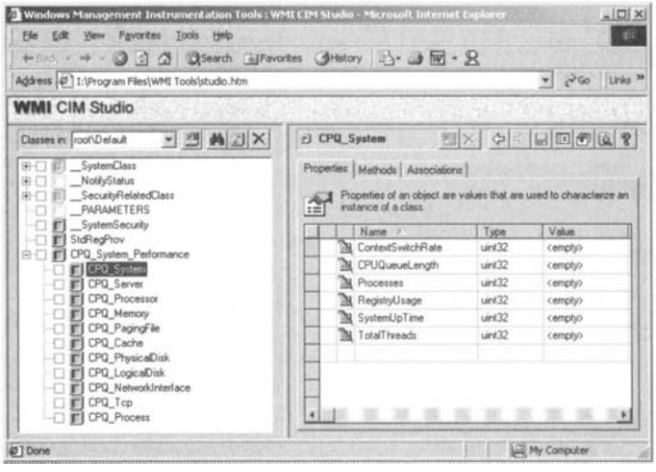

Figure 5.32 *The CPQ_System_Performance superclass with its subclasses, as shown in WMI CIM Studio.*

5.5.2 Microsoft Operation Manager

When Microsoft Operation Manager (MOM) is installed, the WMI provider coming with the product is not installed by default. In the MOM installation directory ("C:\Program Files\Microsoft Operation Manager 2000\One Point"), there is a MOF file called OM.MOF. You must compile the **OM.MOF** file with **MOFComp.Exe** to get the *MOM* provider and its unique extrinsic event class registered in the CIM repository.

Once completed, a new namespace is created, called **Root\MCS**. The *MOM* provider is implemented as an event provider, which means that WQL event queries don't need to use the **WITHIN** statement. The WQL event query will look as follows:

```
Select * From OM_Alert
```

The *OM_Alert* extrinsic event class represents the alerts generated on the MOM console. The aim of this provider with its unique class is to notify WMI event subscribers interested in receiving notifications for the MOM alerts. The *OM_Alert* class exposes the properties listed in Table 5.36.

The biggest interest of the *OM_Alert* extrinsic event class resides in the possibility to interface any other management products with MOM. Of course, this is only possible on the condition that management products support WMI data utilization. For instance, a company can decide to use MOM to manage and monitor the Microsoft environment, while the gen-

Table 5.36 *The OM_Alert Extrinsic Event Class with the MOM SP1 Classes*

MOM Provider classes (RTM)	
Class	**Description**
OM_Alert	One Point Operations Manager Alert
MOM Provider classes (SP1)	
Class	**Description**
MSFT_Alert	Represents a MOM alert and it provides access to details about the alert's status and other properties.
MSFT_AlertHistory	Represents a MOM alert history and it provides access to details about the alert's status and other properties.
MSFT_AlertResolutionState	Represents a MOM alert and it provides access to details about the alert's status and other properties.
MSFT_Computer	Represents a MOM computer in the managed MOM environment.
MSFT_ComputerGroup	Represents a MOM computer group.
MSFT_MicrosoftOperationsManager	Represents information about the MOM software (installed date, version).
MSFT_TodayStatistics	Provides daily sums for several of the most important measurements in the network environment Microsoft Operations Manager is monitoring.
MSFT_Script	Represents a MOM script with all its parameters.
MSFT_AlertHistoryToAlertResolutionState	Associates an alert history with its state.
MSFT_AlertToAlertResolutionState	Associates an alert with its state.
MSFT_AlertToAlertHistory	Associates an alert with history.
MSFT_AlertToComputer	Associates an alert with a computer.
MSFT_ComputerToComputerGroup	Associates a computer with a computer group.
MSFT_ScriptToComputerGroup	Associates a script to a computer group.

erated alerts are forwarded to another enterprise management software having WMI data utilization capabilities. In such a case, the enterprise management software will utilize alerts generated from the *OM_Alert* class.

When the MOM Service Pack 1 is installed, another WMI provider extends the management capabilities and the information set available from MOM through WMI. This new WMI provider exposes alerts, alert history, computers, computer groups, and "computer to computer group" associations. However, by default, as with the previous *MOM* provider, it is not registered by default in the CIM repository

Once the Service Pack 1 is installed, the MOM installation directory ("C:\Program Files\Microsoft Operation Manager 2000\One Point") contains a MOF file called **MOMWMI.MOF**. Once the registration with **MOFCOMP.EXE** is completed, it creates a new namespace called Root\MOM, containing classes listed in Table 5.36.

5.5.3 HP OpenView Operations for Windows

Understanding the WMI capabilities and the WMI information exposed by HP OpenView Operations for Windows v7 (OVOW v7) is invaluable for people who want to develop automated tasks, applications, or scripts integrating with OVOW v7. OVOW v7 provides the first and only service management engine based completely on the WBEM/WMI specification. The use of WMI/WBEM permits extensive customization of the OpenView Operations Management Server via scripting and automation tools—a capability used heavily by lights-out and highly automated enterprises.

There are hundreds of WMI classes, and it is impossible to review all the possibilities offered by the OVOW v7 WMI implementation in a few pages. Most classes available are used by OVOW for internal functions, such as supporting OVOW interconsole synchronization of events, status, and actions. Therefore, external WMI consumers are not officially supported. It is important to understand that even if the features presented in this section are fully functional, HP does not officially support them, because they rely on some internal WMI functions of the OVOW v7 architecture. These WMI internal functions are subject to change without notice with the next release of OVOW.

Despite this restriction, some aspects can be very useful in the field and deserve closer examination. For instance, the application of some typical tasks, such as programmatically acknowledging alerts or monitoring changes to some alerts, demonstrates how OVOW v7 can easily be extended with the use of a standard management interface such as WMI.

Therefore, we will concentrate on a very typical WMI class exposed by the OVOW v7 WMI implementation and used heavily by the OVOW console: the *OV_Message* class.

When installing OVOW v7 in a Windows 2000 system, the setup process creates three new namespaces in the CIM repository:

- **Root\HewlettPackard\OpenView**
- **Root\HewlettPackard\OpenView\Console**
- **Root\HewlettPackard\OpenView\Data**

By default, only administrators have full access to WMI namespaces. However, the OVOW setup modifies the default security on the **Root\HewlettPackard\OpenView\Data** namespace to allow access to members of the Local Administrators, HP-OVE-ADMINS, and HP-OVE-OPERATORS groups.

All namespaces contain one or more WMI providers, as summarized in Table 5.37.

Table 5.37 *The HP OpenView Providers Capabilities*

Provider Name	Class Provider	Instance Provider	Method Provider	Property Provider	Event Provider	Event Consumer Provider	Support Get	Support Put	Support Enumeration	Support Delete
Root\HewlettPackard\OpenView										
storeprov_PermEventConsumer						X				
Root\HewlettPackard\OpenView/Console										
OVConsoleEvents			X		X					
Root\HewlettPackard\OpenView/Data										
AdsProv		X							X	
DnsProv		X							X	
NetProv		X							X	
NnmProv		X							X	
OV_ActionRequest_InstanceProvider		X	X				X	X	X	X
OV_ActionResponseEvent_Provider					X					
OV_Message_InstanceCreationAndDeletionEvent_Provider					X					
OV_Message_InstanceProvider		X	X				X		X	
OV_Message_NumberOfAnnotationsChangeEvent_Provider					X					
OV_Message_SeverityChangeEvent_Provider					X					
OV_Message_StateChangeEvent_Provider					X					
OV_Message_TextChangeEvent_Provider					X					
OV_MessageAction_InstanceProvider		X	X				X			
OV_MessageAction_StateChangeEvent_Provider					X					
OvAutoDeploy_PermEventConsumer						X				
OvEpStatusEventProvider					X					
OvEpStatusInstProvider		X					X		X	
pppprov		X			X		X		X	
RegPropProv				X			X	X		
StoreInstanceProv		X			X		X	X	X	X
storeprov_PermEventConsumer						X				
UnmAgtProv		X							X	

As we can see in Table 5.37, OVOW v7 brings several instance providers, event providers, event consumer providers, method providers, and one property provider. Some providers also combine several roles. Each provider supports a set of WMI classes that mostly represent OVOW manageable objects, such as managed nodes, services, node groups, and console messages.

As an example, we can track OVOW v7 alerts displayed on the OVOW console (see Figure 5.33) to trigger the execution of a script. With the script Sample 6.17 ("A generic script for asynchronous event notification") in the appendix, we can easily track alerts via this WMI alert consumer script. Of course, in this case, we use a script as a WMI consumer, but this can be any other WMI consumer application, such as another Enterprise Management System, able to consume WMI alerts. To track all new messages displayed at the OVOW console, we should start the script with the following WQL event query:

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

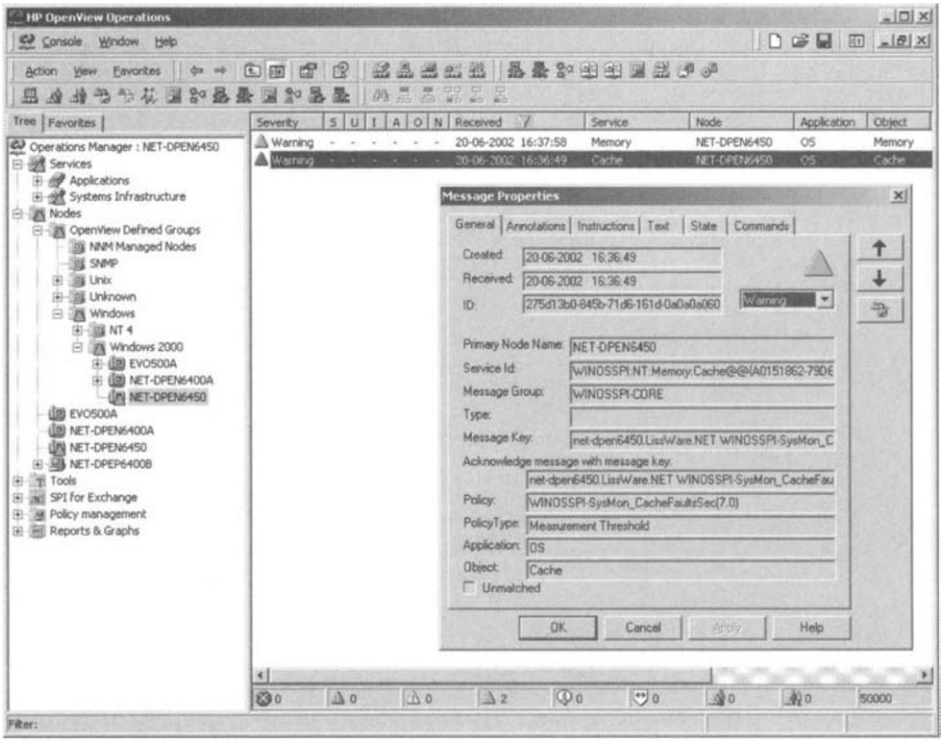

Figure 5.33 *The HP OpenView Operations for Windows console showing alerts.*

The WQL query makes use of the *__InstanceCreationEvent* intrinsic event class, which captures any instance creation event of the *OV_Message* WMI class. The *OV_Message* class is a template representing the OVOW v7 alerts. With this WQL event query, the script receives a notification each time a new alert is available from the OVOW console. By observing the displayed data, you will see that the information exactly matches the alert information shown in Figure 5.33.

```
C:\>GenericEventAsyncConsumer.wsf "Select * From __InstanceCreationEvent
                                   Where TargetInstance ISA 'OV_Message'"
                                   /Namespace:root\HewlettPackard\OpenView\Data
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Waiting for events...

BEGIN - OnObjectReady.
Thursday, 20 June, 2002 at 16:36:49: '__InstanceCreationEvent' has been triggered.
  TargetInstance (wbemCimtypeObject)
    AcknowledgeAfterTroubleTicket = False
    AgentId = 059bbdd0-7eed-71d6-1e0e-0a0a0a060000
    Application = OS
    AutomaticAction = (null)
    ConditionId = 00000000000000000000000000000000
    CreateTroubleTicketInterface = False
    DoNotification = False
    *Id = 275d13b0-845b-71d6-161d-0a0a0a060000
    InstructionAvailable = False
    IsProxied = False
    LogOnly = False
    MessageGroup = WINOSSPI-CORE
    MessageKey = net-dpen6450.LissWare.NET WINOSSPI-SysMon_CacheFaultsSec
    MessageKeyRelation = net-dpen6450.LissWare.NET WINOSSPI-SysMon_CacheFaultsSec
    NodeName = {A0151862-79D6-47F9-8952-97F563F29429}
    NumberOfAnnotations = 0
    Object = Cache
    OperatorAction = (null)
    OriginalId = 00000000000000000000000000000000
    OriginalServiceId = WINOSSPI:NT:Memory:Cache@@<$MSG_NODE_ID>
    OriginalText = Monitor WINOSSPI-SysMon_CacheFaultsSec: Threshold:... Value: 77.17
    ServiceId = WINOSSPI:NT:Memory:Cache@@{A0151862-79D6-47F9-8952-97F563F29429}
    Severity = 4
    Source = WINOSSPI-SysMon_CacheFaultsSec(7.0)
    SourceType = 8
    State = 2
    Text = Object: Memory, Counter: Cache Faults/sec: The threshold of 15 ...
    TimeCreated = 20-06-2002 16:36:49 (20020620163649.000000+120)
    TimeOfStateChange = 20-06-2002 16:36:49 (20020620163649.000000+120)
    TimeReceived  = 20-06-2002 16:36:49 (20020620163649.000000+120)
    Type (wbemCimtypeString) =
    Unmatched (wbemCimtypeBoolean) = False
    UsedNotificationInterfaces (wbemCimtypeString) =
    UserOfStateChange (wbemCimtypeString) =
END - OnObjectReady.
..:
Finished.
```

If we closely examine the *OV_Message* WMI class (), we can see that besides the 34 properties exposed by the class, there are also 16 methods available. Table 5.38 was generated with the **LoadCIMinXL.Wsf** script (see Sample 4.32 in the appendix).

Table 5.38 *The OV_Message Class*

Properties	Syntax	Access	Description
AcknowledgeAfterTroubleTicket	boolean	Read	If trouble ticket generation succeeds, acknowedge the message
AgentId	string	Read	The "Agent ID" that uniquely identifies the agent (different than the "Node ID) - used to support NAT and DHCP environments
Application	string	Read	The "Application" that created the message. Similar in many respects to the Windows Event Log "Source" field.
AutomaticAction	object:OV_MessageAction	Read	If an automatic action is assigned to the event, this object is valid.
ConditionId	string	Read	The "Condition" id (a GUID) of the rule that triggered the event
CreateTroubleTicketInterface	boolean	Read	True if this event should generate a new trouble ticket
DoNotification	boolean	Read	True if this event should generate an e-mail message, SMS or page
Id (key)	string	Read	The internal ID of the message. This is the key for the OV_Message table, and can be used to directly retrieve a message object.
InstructionAvailable	boolean	Read	True if instruction text is available
IsProxied	boolean	Read	True if proxied through a secondary node. Not currently used
LogOnly	boolean	Read	True if the event is to be written to the log (acknowledged) browser without display to the console
MessageGroup	string	Read	A grouping to classify the message - arbitrary
MessageKey	string	Read	A unique "Key" to identify the message. Used to keep only the most recent message displayed in the browser. Also used for "paired" events (good acknowleged bad)
MessageKeyRelation	string	Read	The message key that will be acknowledged when this message is received. Can contain wildcards and pattern matches.
NodeName	string	Read	The ID of the node in the OV_ManagedNode table. This is not the display name of the node, but the GUID representing the node in the OV_ManagedNode table. Query or GetObject from OV_ManagedNode against objMessage.NodeName to get the Caption or PrimaryNodeName values.
NumberOfAnnotations	sint32	Read	Number of available annotations on a message
Object	string	Read	The "Object" of the message. Similar in many respects to the Windows Event Log "Category" field.
OperatorAction	object:OV_MessageAction	Read	Exists if an operator-initiated action is available for this message
OriginalId	string	Read	The original message ID generating this event
OriginalServiceId	string	Read	The original service ID assigned to the event
OriginalText	string	Read	The original text of the event
ServiceId	string	Read	The ID of the service that this message belogs to. This is not the display name of the service, but the internal ID representing the service in the OV_Service table. Query or GetObject from the OV_Service against objMessage.ServiceID to get the Caption or Description properties.
Severity	sint32	Read	Defines the severity of the message ("Unknown=1", "Normal=2", "Warning=4", "Minor=8", "Major=16", "Critical=32")
Source	string	Read	Origin of the message.
SourceType	sint32	Read	Defines the message source type ("Unknown=0x2000", "MPE Console=0x2001", "Open Message Interface=0x2002", "Logfile Entry | Windows Event Log=0x2004", "Measurement Threshold=0x2008", "SNMP Interceptor=0x2010", "Message Stream Interface=0x2020", "Reserved=0x2040", "Reserved=0x2080", "Scheduled Command=0x2100", "Measurement Threshold=0x2200", "Windows Management Interface=0x2400")
State	sint32	Read	Defines the state of the message ("Undefined=1", "Unowned=2", "Owned=3", "Acknowledged=4", "Node Deleted=5", "Deleted=6")
Text	string	Read	The text of the message as displayed to the user.
TimeCreated	datetime	Read	The time the event was created at the node
TimeOfStateChange	datetime	Read	The time when the state of the message was last changed (owned, disowned, acknowledged)
TimeReceived	datetime	Read	The time the event was received at the management server
Type	string	Read	The internal type of the message, assigned by the policy creator. This is often used in Manager of Manager environments to selectively route messages around.
Unmatched	boolean	Read	If the event didn't match any conditions of the policy, but the "send unmatched" option was enabled on the node.
UsedNotificationInterfaces	string	Read	Indicates if the notification interface was used
UserOfStateChange	string	Read	The NT/AD username of the user that performed the last state change (own,disown, acknowledge, unacknowledge)

Table 5.38 *The OV_Message Class (continued)*

Methods	Parameters	Syntax	Description
GetInstruction	**Output:**		Gets the instruction text for this event.
	Instruction	string	
	ReturnValue	sint32	
ChangeText	**Input:**		Sets new text for an existing message, and fires a TextChange event
	NewText	string	
	Output:		
	ReturnValue	sint32	
ChangeSeverity	**Input:**		Modifies the message severity ("Unknown", "Normal", "Warning", "Minor", "Major", "Critical")
	NewSeverity	sint32	
	Output:		
	ReturnValue	sint32	
Acknowledge	**Output:**		Marks the message as acknowledged (resolved/superceded)
	ReturnValue	sint32	
Unacknowledge	**Output:**		Marks the message as unacknowledged (unresolved problem report)
	ReturnValue	sint32	
Own	**Output:**		Marks the message as "Owned". The caller of the method is the owner.
	ReturnValue	sint32	
Disown	**Output:**		Marks the messages as "Unowned".
	ReturnValue	sint32	
GetAnnotation	**Input:**		Gets a textual annotation to the message. Annotations are available to all operators that can view the message. OV_Message can have any number of annotations attached. It serves as a useful mechanism of tracking updates and changes of an ongoing problem report.
	AnnotationNumber	sint32	
	Output:		
	Annotation	object:OV_MessageAnnotation	
	ReturnValue	sint32	
ModifyAnnotation	**Input:**		Modifies a textual annotation to the message.
	AnnotationNumber	sint32	
	NewText	string	
	Output:		
	ReturnValue	sint32	
DeleteAnnotation	**Input:**		Deletes a textual annotation to the message.
	AnnotationNumber	sint32	
	Output:		
	ReturnValue	sint32	
AddAnnotation	**Input:**		Adds a textual annotation to the message.
	Text	string	
	Output:		
	ReturnValue	sint32	
AcknowledgeMessages	**Input:**		Marks a collection of messages as acknowledged (resolved/superceded)
	MessageIDs	string	
	Output:		
	MassOperationResults	object:OV_Message_MassOperationResult	
	ReturnValue	sint32	
UnacknowledgeMessages	**Input:**		Unmarks a collection of messages as unacknowledged (unresolved problem report)
	MessageIDs	string	
	Output:		
	MassOperationResults	object:OV_Message_MassOperationResult	
	ReturnValue	sint32	
OwnMessages	**Input:**		Marks a collection of messages as "Owned". The caller of the method is the owner.
	MessageIDs	string	
	Output:		
	MassOperationResults	object:OV_Message_MassOperationResult	
	ReturnValue	sint32	
DisownMessages	**Input:**		Marks a collection of messages as "Unowned".
	MessageIDs	string	
	Output:		
	MassOperationResults	object:OV_Message_MassOperationResult	
	ReturnValue	sint32	
CountMessages	**Input:**		For future use, not currently supported.
	WhereClause	string	
	Output:		
	Count	sint32	
	ReturnValue	sint32	

The methods exactly match the selection we get by right-clicking on the alerts in the OVOW console. For example, some of these methods are *Acknowledge*, *AcknowledgeMessages*, *Disown*, *DisownMessages*, *Own*, *OwnMessages*, *Unacknowledge*, *UnacknowledgeMessages*, *AddAnnotation*, and *ChangeSeverity*. In the same way, by using another set of WMI classes, such as *OV_Message_SeverityChangeEvent*, *OV_Message_StateChangeEvent*,

OV_Message_TextChangeEvent, *OV_Message_NumberOfAnnotationsChangeEvent*, and *OV_MessageAction_StateChangeEvent*, it is possible to track the alert state modifications. These extrinsic event classes are very easy to use, since you just need to run the **GenericEventAsyncConsumer.wsf** script with a different WQL event query. For example, to track the alert severity changes, you use the *OV_Message_SeverityChangeEvent* event class in the following WQL event query:

```
Select * From OV_Message_SeverityChangeEvent
```

Since the *OV_Message* class, with its related *OV_Message_InstanceProvider* WMI provider, is used to implement the features to manage the OVOW v7 console messages, it is possible to create a script managing messages from the command line.

Before digging into the code details, let's see the command-line parameters exposed by the script. Basically, the script exposes most methods supported by the *OV_Message* class as a command-line parameter.

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

Usage: OVOWMessageManager.wsf MessageID /Action:value
                              [/AnnotationNumber:value] [/Annotation:value]
                              [/Severity:value] [/Text:value] [/WQLQuery:value]
                              [/Machine:value] [/User:value] [/Password:value]
Options:

MessageID        : OVOW Message(s) ID.
Action           : Determines the OVOW Message action to perform.
AnnotationNumber : Specifies the annotation number.
Annotation       : Specifies the annotation text.
Severity         : Specifies the message severity.
Text             : Specifies the text message. (Used with ChangeText)
WQLQuery         : Specifies WQL Query for the messages to list.
Machine          : Determines the WMI system to connect to.
User             : Determines the UserID to perform the remote connection.
Password         : Determines the password to perform the remote connection.
Examples:

      OVOWMessageManager.wsf /Action:List
      OVOWMessageManager.wsf /Action:List
                             /WQLQuery:"Select * From OV_Message Where Severity=4"
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:View
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Acknowledge
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,
                             098fd400-7ef0-71d6-0b88-0a0a0a060000
                             /Action:AcknowledgeMessages
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                             /Action:Unacknowledge
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,
                             098fd400-7ef0-71d6-0b88-0a0a0a060000
                             /Action:UnacknowledgeMessages
      OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Own
```

```
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,
                       098fd400-7ef0-71d6-0b88-0a0a0a060000
                       /Action:OwnMessages
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:Disown
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,
                       098fd400-7ef0-71d6-0b88-0a0a0a060000
                       /Action:DisownMessages
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:AddAnnotation
                       /Annotation:"This is my message annotation"
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:GetAnnotation /AnnotationNumber:1
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ModifyAnnotation
                       /AnnotationNumber:1
                       /Annotation:"This is my updated message annotation"
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:DeleteAnnotation
                       /AnnotationNumber:1
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Unknown
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Normal
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Warning
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Minor
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Major
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeSeverity /Severity:Critical
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:ChangeText /Text:"This is my new text"
OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                       /Action:GetInstruction
```

Although there are 16 methods exposed by the *OV_Message* class, we will see that most methods use the same scripting techniques. As shown in Sample 5.16, the first lines are devoted to the external functions inclusion (lines 63 through 69), constants declaration (lines 79 through 81), XML command-line parameters parsing (lines 130 through 213), and the WMI connection (lines 225 through 229).

Sample 5.16 *The script initialization phase*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
..:
61:    </runtime>
62:
63:    <script language="VBScript" src=".\Functions\DecodeOVOWManagedNodeFunction.vbs" />
```

```
 64:    <script language="VBScript" src=".\Functions\DecodeOVOWMessageFunction.vbs" />
 65:    <script language="VBScript" src=".\Functions\OVOWErrorHandler.vbs" />
 66:    <script language="VBScript" src=".\Functions\ConvertStringInArrayFunction.vbs" />
 67:    <script language="VBScript" src=".\Functions\DisplayFormattedPropertiesFunction.vbs" />
 68:    <script language="VBScript" src=".\Functions\DisplayFormattedPropertyFunction.vbs" />
 69:    <script language="VBScript" src=".\Functions\TinyErrorHandler.vbs" />
 70:
 71:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
 72:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
 73:
 74:    <script language="VBscript">
 75:    <![CDATA[
 ..:
 79:    Const cComputerName = "LocalHost"
 80:    Const cWMINameSpace = "Root/HewlettPackard/OpenView/Data"
 81:    Const cWMIClass = "OV_Message"
...:
129:    ' -- COMMAND LINE PARSING ------------------------------------------------------------
130:    If WScript.Arguments.Named.Count Then
131:       Select Case Ucase(WScript.Arguments.Named("Action"))
132:              Case "LIST"
133:                   strWQLQuery = WScript.Arguments.Named("WQLQuery")
134:                   If Len(strWQLQuery) = 0 Then
135:                      strWQLQuery = "Select * From " & cWMIClass
136:                   End If
137:                   boolList = True
138:              Case "VIEW"
139:                   boolView = True
...:
203:       End Select
204:    End If
205:
206:    If boolList = False Then
207:       If WScript.Arguments.Unnamed.Count = 0 Then
208:          WScript.Arguments.ShowUsage()
209:          WScript.Quit
210:       Else
211:          arrayWMIOVMessages = ConvertStringInArray (WScript.Arguments.Unnamed.Item(0), ",")
212:       End If
213:    End If
...:
224:    ' -- WMI CONNECTION ---------------------------------------------------------------
225:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
226:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
227:
228:    Set objWMIServices = objWMILocator.ConnectServer(strComputerName, cWMINameSpace, _
229:                                                     strUserID, strPassword)
...:
...:
...:
```

Once the initialization phase is completed, based on the command-line parameters given, a specific section of the script will be executed. Let's start with the **/Action:List** switch. Basically, this switch displays the OVOW v7 messages in the same way as the OVOW console. This switch produces the following output (the output is divided into two blocks, since it is larger than the page).

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

1269 message(s) to list.

OVOW Messages
================================================================================================== →
                                                                                                   →
 Severity        State             Time received                  Object            Application   →
-------------------------------------------------------------------------------------------------- →
   Normal Acknowledged 20020613190429.000000+120                    None HP OpenView Operations   →
  Warning      Unowned 20020613190727.000000+120 opcmona (Monitor Agent) HP OpenView Operations   →
  Warning Acknowledged 20020613191226.000000+120                  Memory                     OS   →
   Normal Acknowledged 20020613191319.000000+120               OvDnsDscr HP OpenView Operations   →
   Normal Acknowledged 20020614090237.000000+120                    None HP OpenView Operations   →
                                                                                                   →
→ ================================================================================
→
→                 Node name        OS Version                              Message ID
→ --------------------------------------------------------------------------------
→ NET-DPEN6450.LissWare.NET Windows 2000(5.0) 9f56fcd0-7eef-71d6-05bf-0a0a0a060000
→ NET-DPEN6450.LissWare.NET Windows 2000(5.0) 098fd400-7ef0-71d6-0b88-0a0a0a060000
→ NET-DPEN6450.LissWare.NET Windows 2000(5.0) bc12ae40-7ef0-71d6-0b88-0a0a0a060000
→ NET-DPEN6450.LissWare.NET Windows 2000(5.0) db5686a0-7ef0-71d6-05bf-0a0a0a060000
→ NET-DPEN6450.LissWare.NET Windows 2000(5.0) 6b8258d0-7f64-71d6-11fd-0a0a0a060000
```

We recognize the different characteristics (severity, state, time received, etc.) of an OVOW message, as shown in Figure 5.33. Sample 5.17 shows how to code the logic retrieving the OVOW message list.

Sample 5.17 *Retrieving a collection of OV_Message instances*

```
...:
...:
...:
232:    ' -- LIST ------------------------------------------------------------------------
233:    If boolList Then
234:       Set objWMIOVMessages = objWMIServices.ExecQuery (strWQLQuery)
...:
237:       WScript.Echo objWMIOVMessages.Count & " message(s) to list." & vbCRLF
238:
239:       If objWMIOVMessages.Count Then
240:          WScript.Echo "OVOW Messages "  & String (193, "=") & vbCRLF
241:          WScript.Echo "       Severity           State" & _
242:                       "             Time received" & _
243:                       "                                  Object" & _
244:                       "                        Application" & _
245:                       "                          Node name" & _
246:                       "           OS Version" & _
247:                       "                                     Message ID"
248:          WScript.Echo String (207, "-")
249:
250:          For Each objWMIOVMessage In objWMIOVMessages
251:              strSeverity = DecodeOVOWMessageSeverity (objWMIOVMessage.Severity) & _
252:                            "(" & objWMIOVMessage.Severity & ")"
253:              strState = DecodeOVOWMessageState (objWMIOVMessage.State) & _
254:                            "(" & objWMIOVMessage.State & ")"
```

```
255:
256:            Set objWMIManagedNode = objWMIServices.Get ("OV_ManagedNode.Name='" & _
257:                                                        objWMIOVMessage.NodeName & "'")
258:            If Err.Number Then
259:               strPrimaryNodeName = "<ERROR>"
260:            Else
261:               strPrimaryNodeName = objWMIManagedNode.PrimaryNodeName
262:               strOSType = DecodeOVOWManagedNodeOSType (objWMIManagedNode.OSType) & "(" & _
263:                           DecodeOVOWManagedNodeOSVersion (objWMIManagedNode.OSVersion) & ")"
264:            End If
265:
266:            WScript.Echo String (15 - Len (strSeverity), " ") & _
267:                         strSeverity & " " & _
...:
280:                         String (36 - Len (objWMIOVMessage.Id), " ") & _
281:                         objWMIOVMessage.Id
282:
283:            Set objWMIManagedNode = Nothing
284:         Next
285:      Else
286:         WScript.Echo "No information available." & vbCRLF
287:      End If
...:
290:   End If
291:
...:
...:
...:
```

The first operation consists of retrieving a collection of instances by executing a WQL data query (line 234). The WQL data query is defined during the command parsing processing (lines 133 through 137 in Sample 5.16). The default WQL data query used is (line 135):

```
Select * From OV_Message
```

Of course, it is possible to specify another data query by using the **/WQLQuery:** switch on the command line (line 133). For example, a valid command line would be:

```
C:\>OVOWMessageManager.wsf /Action:List /WQLQuery:"Select * From OV_Message Where Severity=4"
```

where the severity number corresponds to the severity level listed in Table 5.39.

Once the WQL data query is completed, the script displays a header (lines 240 through 248) and enumerates all instances of the *OV_Message* class available for the specified WQL data query (lines 250 through 284). For each instance listed, Sample 5.17 retrieves and displays a specific set of properties:

- **The Severity level** (lines 251 and 252) is decoded with the DecodeOVOWMessageSeverity() function included at line 64. The DecodeO-

Table 5.39 *The OV_Message and OV_ManagedNode Property Value Meanings*

Severity	
Unknown	1
Normal	2
Warning	4
Minor	8
Major	16
Critical	32

State	
Undefined	1
Unowned	2
Owned	3
Acknowledged	4
Node Deleted	5
Deleted	6

SystemType	
Alpha Family	35
Other	1
PA-RISC Family	144
Pentium Compatible	11
Power PC Family	27
SPARC Family	80

SourceType	
Unknown	0x2000
MPE Console	0x2001
Open Message Interface	0x2002
Logfile Entry | Windows Event Log	0x2004
Measurement Threshold	0x2008
SNMP Interceptor	0x2010
Message Stream Interface	0x2020
Reserved	0x2040
Reserved	0x2080
Scheduled Command	0x2100
Measurement Threshold	0x2200
Windows Management Interface	0x2400

OSType	
AIX	9
HPUX	8
LINUX	36
SNMP	65535
Solaris	29
Tru64	6
Unknown	0
Windows 2000	58
Windows NT	18
Windows XP	101
Windows Server 2003	102

OSVersion	
4.0	18000
2.6	29010
7	29020
8	29030
Red Hat 6.X	36206
Red Hat 7.X	36207
SuSE 6.X	36306
SuSE 7.X	36307
Turbo 6.X	36406
Turbo 7.X	36407
5.0	58000
V1	65535000

VOWMessageSeverity() function uses the information shown in Table 5.39 to determine the corresponding severity text.

- **The Message State** (lines 253 and 254) is decoded with the DecodeOVOWMessageState() function, also included at line 64. The DecodeOVOWMessageState() function uses information from Table 5.39 to determine the corresponding state text.
- **The Managed Node instance** (lines 256 and 257) is retrieved by requesting an instance of the *OV_ManagedNode* class with the node identifier exposed by the *NodeName* property of the *OV_Message* class. An *OV_ManagedNode* instance represents a managed node (i.e., computer) in the OVOW world. The node identifier is nothing other than a GUID number generated by OVOW v7 for each node configured or discovered in OVOW.
- **The Managed Node name** (line 261) corresponds to the DNS or WINS node name and is available from the *PrimaryNodeName* property exposed by the *OV_ManagedNode* class.
- **The Managed Node OS type and version** (lines 262 and 263) are, respectively, decoded with the DecodeOVOWManagedNodeOSType() and DecodeOVOWManagedNodeOSVersion() functions included at line 63. The DecodeOVOWManagedNodeOSType() and DecodeOVOWManagedNodeOSVersion() functions use the

information in Table 5.39 to determine the text corresponding to the OS type and version values.

- **The Message ID** (lines 280 and 281) is the unique identifier of the message. Its value is a GUID number generated by OVOW. The script retrieves the message identifier by reading the ID property value exposed by the *OV_Message* class. The message identifier will be required to manage the various message attributes (*severity*, *state*, *annotation*, etc.)

From line 266 through 281, the script displays all these properties in a series of formatted columns.

It is possible to see all properties of an OVOW console message by referencing its GUID number (visible with the **/Action:List** switch) and the use of the **/Action:View** switch. For example, the following command line will show all properties of the OVOW message using the message identifier: 5bc2e7a0-7f66-71d6-09ae-0a0a0a060000:

```
C:\>OVOWMessageManager.wsf 5bc2e7a0-7f66-71d6-09ae-0a0a0a060000 /Action:View
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

AcknowledgeAfterTroubleTicket: ..... FALSE
AgentId: ........................... 059bbdd0-7eed-71d6-1e0e-0a0a0a060000
Application: ....................... OS
ConditionId: ....................... 00000000000000000000000000000000000000
CreateTroubleTicketInterface: ..... FALSE
DoNotification: .................... FALSE
*Id: ............................... 5bc2e7a0-7f66-71d6-09ae-0a0a0a060000
InstructionAvailable: .............. FALSE
IsProxied: ......................... FALSE
LogOnly: ........................... FALSE
MessageGroup: ...................... WINOSSPI-CORE
MessageKey: ........................ net-dpen6450.LissWare.NET
                                     WINOSSPI-SysMon_CacheFaultsSec
MessageKeyRelation: ................ net-dpen6450.LissWare.NET
                                     WINOSSPI-SysMon_CacheFaultsSec
NodeName: .......................... {A0151862-79D6-47F9-8952-97F563F29429}
NumberOfAnnotations: ............... 1
Object: ............................ Cache
OriginalId: ........................ 00000000000000000000000000000000000000
OriginalServiceId: ................. WINOSSPI:NT:Memory:Cache@@<$MSG_NODE_ID>
OriginalText: ...................... Monitor WINOSSPI-SysMon_CacheFaultsSec:
                                     Threshold: Scripting Value: 26.03
ServiceId: ......................... WINOSSPI:NT:Memory:Cache@@{A0151862-79D6-...9429}
Severity: .......................... Critical
Source: ............................ WINOSSPI-SysMon_CacheFaultsSec(7.0)
SourceType: ........................ 8
State: ............................. Acknowledged
Text: .............................. Object: Memory, Counter: Cache Faults/sec:
                                     The threshold of 20 has been crossed at 26!
TimeCreated: ....................... 20020614091425.000000+120
TimeOfStateChange: ................. 20020614091934.000000+120
TimeReceived: ...................... 20020614091425.000000+120
```

```
34:   Type: ..................................
35:   Unmatched: ............................. FALSE
36:   UsedNotificationInterfaces: ..............
37:   UserOfStateChange: ...................... SYSTEM
```

The logic to display this information is not really different from the logic used for the **/Action:List** switch, and the output is similar to the one obtained with the **GenericEventAsyncConsumer.wsf** script used previously. This is illustrated in Sample 5.18.

Sample 5.18 *Retrieving all properties of an OV_Message instance*

```
...:
...:
...:
291:
292:    ' -- VIEW ------------------------------------------------------------------------
293:    If boolView Then
294:       Set objWMIOVMessage = objWMIServices.Get (cWMIClass & "='" & _
295:                                                 arrayWMIOVMessages (0) & "'")
...:
298:       Set objWMIPropertySet = objWMIOVMessage.Properties_
299:       For Each objWMIProperty In objWMIPropertySet
300:           Select Case objWMIProperty.Name
301:                  Case "AutomaticAction"
302:                       If IsObject (objWMIProperty.Value) Then
303:                          DisplayFormattedProperty objWMIOVMessage, _
304:                                 objWMIProperty.Name, _
305:                                 "Automatic Action", _
306:                                 Null
307:
308:                          Set objWMIMessageAction = objWMIProperty.Value
309:                          DisplayFormattedProperties objWMIMessageAction, 2
310:                          Set objWMIMessageAction = Nothing
311:                          WScript.echo
312:                       End If
313:                  Case "OperatorAction"
314:                       If IsObject (objWMIProperty.Value) Then
315:                          DisplayFormattedProperty objWMIOVMessage, _
316:                                 objWMIProperty.Name, _
317:                                 "Operator Action", _
318:                                 Null
319:
320:                          Set objWMIMessageAction = objWMIProperty.Value
321:                          DisplayFormattedProperties objWMIMessageAction, 2
322:                          Set objWMIMessageAction = Nothing
323:                          WScript.echo
324:                       End If
325:                  Case "Severity"
326:                       DisplayFormattedProperty objWMIOVMessage, _
327:                              objWMIProperty.Name, _
328:                              DecodeOVOWMessageSeverity (objWMIOVMessage.Severity), _
329:                              Null
330:                  Case "SourceType"
331:                       DisplayFormattedProperty objWMIOVMessage, _
332:                              objWMIProperty.Name, _
333:                              DecodeOVOWMessageSourceType (objWMIOVMessage.SourceType), _
```

```
334:                              Null
335:                    Case "State"
336:                         DisplayFormattedProperty objWMIOVMessage, _
337:                              objWMIProperty.Name, _
338:                              DecodeOVOWMessageState (objWMIOVMessage.State), _
339:                              Null
340:                    Case Else
341:                         DisplayFormattedProperty objWMIOVMessage, _
342:                              objWMIProperty.Name, _
343:                              objWMIProperty.Name, _
344:                              Null
345:               End Select
346:          Next
347:          Set objWMIPropertySet = Nothing
348:
349:          Set objWMIOVMessage = Nothing
350:     End If
351:
...:
...:
...:
```

First, the script retrieves the instance corresponding to the message ID given on the command line (lines 294 and 295). Since we can specify several message IDs separated by a column on the command line, the script converts all message IDs given in an array. This operation is completed during the command-line parsing at line 211 in Sample 5.16. This is why line 295 refers to the element zero of the array. Once done, the script enumerates all properties available from the *OV_Message* class. If a property requires a specific decoding, the **Select Case** statement will handle that for each property listed in the case. For example, we will recognize the decoding functions used previously for the severity and state properties. The *SourceType* property follows the same logic and is decoded by invoking the DecodeOVOWMessageSourceType() included at line 64. The *AutomaticAction* and *OperatorAction* properties are slightly different, since they expose an instance of the *OV_MessageAction* class. Instances of this class expose information about actions that must be performed in relation to the console message (automatic or operator actions). Since an *OV_MessageAction* instance is contained in these two properties, another set of properties is available. Therefore, the script invokes the DisplayFormattedProperties() function to display all its properties. Behind the scenes, the DisplayFormattedProperties() invokes the DisplayFormattedProperty() function to display the property values. These two functions are included at lines 67 and 68, respectively.

As previously mentioned, the *OV_Message* class also exposes a series of methods allowing the management of OVOW messages. From a scripting point of view, we can classify the methods into two categories:

- The methods related to one specific instance of an OVOW message. These methods are: *Acknowledge*, *Unacknowledge*, *Own*, *Disown*, *AddAnnotation*, *GetAnnotation*, *ModifyAnnotation*, *DeleteAnnotation*, *ChangeSeverity*, *ChangeText*, *GetInstruction*, and *CountMessages*.
- The methods not specifically related to one particular OVOW message but to a series of OVOW messages. These methods are: *AcknowledgeMessages*, *UnacknowledgeMessages*, *OwnMessages*, and *DisownMessages*.

As with the **/Action:View** command-line parameter, the management of a specific *OV_Message* instance starts with the retrieval of the *OV_Message* instance corresponding to the message ID specified on the command line. Some command-line examples managing an OVOW message are:

```
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Acknowledge
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Unacknowledge
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Own
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000 /Action:Disown
```

The scripting technique is always the same for any of these command-line parameters. Sample 5.19 shows the code used to acknowledge an OVOW message. Only the invoked method differs if you want to unacknowledge, own, or disown an OVOW message.

Sample 5.19 *Managing a specific OV_Message instance*

```
...:
...:
...:
352:    ' -- ACKNOWLEDGE ------------------------------------------------------------------
353:    If boolAcknowledge Then
354:       Set objWMIOVMessage = objWMIServices.Get (cWMIClass & "='" & _
355:                                                 arrayWMIOVMessages (0) & "'")
...:
358:       intRC = objWMIOVMessage.Acknowledge()
359:       If intRC Then
360:          OVOWErrorHandler (intRC)
361:       Else
362:          WScript.Echo "Message acknowledge successfully completed."
363:       End If
...:
366:    End If
367:
...:
...:
...:
```

The technique is pretty easy. The first step retrieves an instance of the OVOW message using the message ID given on the command line (lines 354 and 355). Next, the corresponding *OV_Message* method is invoked

(line 358). If an error occurs, the OVOWErrorHandler() function is invoked to display the corresponding error message. This function is included at line 65.

Other methods, such as *AddAnnotation*, *GetAnnotation*, *ModifyAnnotation*, *DeleteAnnotation*, *ChangeSeverity*, *ChangeText*, *GetInstruction*, and *CountMessages*, follow the same logic. However, these methods require some extra parameters. For example, to change the message severity, the **/Action:ChangeSeverity** switch must be given with the **/Severity:Minor** switch to specify the severity level (see Sample 5.20). The severity level is converted to its corresponding value (see Table 5.39) during the command-line parsing parameters. An example command line would be:

```
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                           /Action:ChangeSeverity /Severity:Minor
```

Sample 5.20 *Changing the OVOW message severity level*

```
...:
...:
...:
547:
548:    ' -- CHANGE SEVERITY ------------------------------------------------------------------
549:    If boolChangeSeverity Then
550:       Set objWMIOVMessage = objWMIServices.Get (cWMIClass & "='" & _
551:                                                 arrayWMIOVMessages (0) & "'")
...:
554:       intRC = objWMIOVMessage.ChangeSeverity(intSeverity)
555:       If intRC Then
556:          OVOWErrorHandler (intRC)
557:       Else
558:          WScript.Echo "Message ChangeSeverity successfully completed."
559:       End If
...:
562:    End If
563:
...:
...:
...:
```

After retrieving the *OV_Message* instance corresponding to the message ID given on the command line (lines 550 and 551), the script invokes the *ChangeSeverity* method with the desired severity level (line 554) input parameter. Next, the error handling is processed, as previously.

A final peculiarity in the scripting technique concerns the *GetAnnotation* method invocation—not especially in the command-line parameters, such as:

```
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000
                           /Action:GetAnnotation /AnnotationNumber:1
```

but more in the way that the annotation is returned by WMI. Sample 5.21 shows this.

Sample 5.21 *Retrieving the OVOW message annotation*

```
...:
...:
...:
491:
492:    ' -- GET ANNOTATION ------------------------------------------------------------------
493:    If boolGetAnnotation Then
494:       Set objWMIOVMessage = objWMIServices.Get (cWMIClass & "='" & _
495:                                                 arrayWMIOVMessages (0) & "'")
...:
498:       intRC = objWMIOVMessage.GetAnnotation(intAnnotationNumber, _
499:                                             objWMIMessageAnnotation)
500:       If intRC Then
501:          OVOWErrorHandler (intRC)
502:       Else
503:          If Len (objWMIMessageAnnotation.Text) Then
504:             WScript.Echo "Message annotation: " & vbCRLF & vbCRLF & _
505:                          objWMIMessageAnnotation.Text & vbCRLF
506:          Else
507:             WScript.Echo "Message Annotation not available." & vbCRLF
508:          End If
509:          WScript.Echo "Message GetAnnotation successfully completed."
510:       End If
...:
513:    End If
514:
...:
...:
...:
```

Once the OVOW message instance is retrieved (lines 494 and 495), the *GetAnnotation* method is invoked (lines 498 and 499). This method requires two parameters: one input parameter, which corresponds to the annotation number, and one output parameter, which is returned as an instance of the *OV_MessageAnnotation* class. Once this instance is retrieved, the script checks that the annotation text is not null by examining the length of the text property (line 503) and displaying the annotation text (lines 504 and 505) if it is available.

Instead of managing OVOW messages one by one, it is possible to acknowledge, unacknowledge, own, or disown a collection of messages. Some command-line examples are:

```
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,098fd400-7ef0-71d6-0b88-0a0a0a060000
                           /Action:AcknowledgeMessages
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,098fd400-7ef0-71d6-0b88-0a0a0a060000
                           /Action:UnacknowledgeMessages
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,098fd400-7ef0-71d6-0b88-0a0a0a060000
                           /Action:OwnMessages
```

```
C:\>OVOWMessageManager.wsf 60ad8ae0-9014-71d6-08a8-0a0a0a060000,098fd400-7ef0-71d6-0b88-0a0a0a060000
                           /Action:DisownMessages
```

Because these *OV_Message* invoked methods do not refer to a particular OVOW message, the methods performing these actions are implemented as a static method (*static* qualifier set to true). Therefore, these methods must be invoked from an instance of a class (which means in this case an instance of the *OV_Message* class) and not from an instance of a real manageable entity (which means in this case, an instance of the *OV_Message* class created with a message ID). Sample 5.22 shows how to proceed.

Sample 5.22 *Managing a series of OV_Message instances*

```
...:
...:
...:
367:
368:    ' -- ACKNOWLEDGE MESSAGES ------------------------------------------------------------
369:    If boolAcknowledgeMessages Then
370:       Set objWMIClass = objWMIServices.Get (cWMIClass)
...:
373:       intRC = objWMIClass.AcknowledgeMessages(arrayWMIOVMessages)
374:       If intRC Then
375:          OVOWErrorHandler (intRC)
376:       Else
377:          WScript.Echo "Messages acknowledge successfully completed."
378:       End If
...:
381:    End If
382:
...:
...:
...:
```

Instead of retrieving an instance corresponding to an OVOW message ID, the script retrieves an instance of the class only (line 370). Next, it invokes the *AcknowledgeMessages* method, which requires an array containing all message IDs of the OVOW messages that must be acknowledged (line 373). During the command-line parameters parsing, all message IDs given on the command line (and separated by a column) are stored in an array (line 211 in Sample 5.16). This technique is exactly the same for the *UnacknowledgeMessages*, *OwnMessages*, and *DisownMessages* methods.

5.6 WMI and the .NET Framework

The .NET Framework programming environment is an extensive subject. Even if we only consider the relationship that exists between WMI and the .NET Framework, this topic represents enough information to at least ded-

icate an entire chapter in this book. However, in this book, we have considered the WMI scripting aspect without considering the programming aspects from languages such as C, C++, or Visual BASIC. Looking at the .NET Framework implies that we leave the scripting world to enter the world of programming with languages such as C# or VB.NET. Therefore, we will just give a brief overview of the .NET Framework WMI capabilities, since this topic is beyond the scope of this book. Some basic knowledge about the .NET Framework architecture, its concepts, and its usage from **Visual Studio.NET** are required.

5.6.1 The .NET Framework WMI information access

In this book, we have exclusively used the **SWbemScripting** COM objects to access WMI information from scripts. However, from C or C++ applications, it is also possible to access the same WMI information by using the native WMI COM interfaces, known as the **IWbem** interfaces. Once the .NET Framework is part of the system, another technology is available to applications accessing WMI information. The .NET WMI object model exposed by the *System.Management* and the *System.Management.Instrumentation* .NET namespaces are designed to support the utilization and the providing of WMI information:

- **The *System.Management* namespace:** This .NET namespace contains .NET classes that provide access to a rich set of management information and management events about the system, devices, and applications instrumented to the WMI. Applications and services can query for interesting management information, using .NET classes derived from *ManagementObjectSearcher* and *ManagementQuery* classes, or subscribe to a variety of management events using the *ManagementEventWatcher* class. Simply said, the *System.Management* namespace offers the required functionality for the WMI clients to access management data.
- **The *System.Management.Instrumentation* namespace:** This .NET namespace provides the .NET classes necessary to instrument applications for management and expose their management information and events through WMI to potential consumers. Consumers can then easily manage, monitor, and configure these applications acting as WMI information providers. The management data is available for WMI scripts or any other applications. Simply said, the *System.Management.Instrumentation* namespace offers the required functionality for WMI applications to provide management information (managed

code) to potential WMI clients. It offers an alternate route for developing WMI providers to the C and C++ COM interfaces.

These two .NET namespaces are totally independent from each other. For instance, if an application provides information with the *System.Management.Instrumentation* namespace, this does not limit the visibility of the instrumentation to the *System.Management* classes, which means that the information provided can be utilized by any other WMI information types (i.e., WMI COM API). On the other hand, if you use the *System.Management* classes, this will allow you to access any WMI information, not just the instrumentation written using the *System.Management.Instrumentation* namespace.

Using the .NET Framework technology to access WMI information implies that we use a programming object model different from the one used for scripting. Does it mean that all material we learned before is not applicable anymore? Well, from a coding technique point of view, it is clear that a completely new object model must be relearned, since it imposes a new coding technique besides the new .NET languages adaptations. However, since the .NET Framework acts as a WMI consumer relying on the existing WMI infrastructure (i.e., WMI COM objects, CIMOM, CIM repository), all WMI classes and information provided under Windows NT, Windows 2000, Windows Server 2003, and Windows XP remain valid. This is where we clearly see the advantage of the abstraction layer that the CIM repository and the CIM Object Manager provide!

Besides the new WMI .NET object model, version 1.00 of the .NET Framework acted as a WMI provider as well. It brought a new WMI provider, called *NetFrameworkv1Provider* provider. Known as the Configuration provider, it supported a collection of WMI classes to manage the .NET Framework and its components. However, subsequent versions of the .NET Framework do not include this provider anymore.

Basically, the .NET Framework is an addition to the existing WMI infrastructure. It is an add-on from a WMI application programming point of view (to develop WMI consumers and providers) and from a .NET Framework manageability point of view (since it brings a WMI provider to manage the .NET application configuration settings).

Figure 5.34 shows the three WMI tiers and identifies how the *System.Management* namespace is layered on WMI: Microsoft Windows Forms (Windows Forms), Web Forms (ASP.NET), and management applications can act as clients that access the WMI instrumentation. At the same time, management providers can be either existing codes, which wrap system or

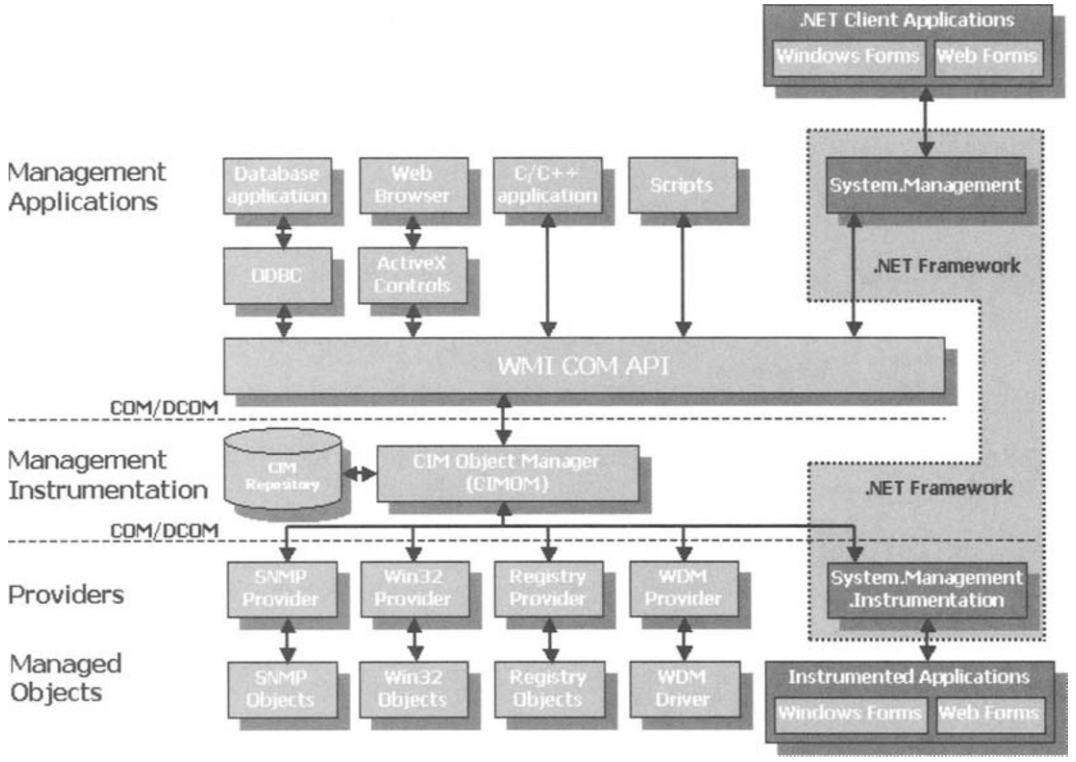

Figure 5.34 *The .NET Framework and the WMI architecture.*

application instrumentation, or Windows Forms and Web Forms/ASP.NET applications, which expose management instrumentation about themselves to other clients by using the *System.Management.Instrumentation* namespace.

In this section, we will only cover the WMI information utilization aspect supported by the *System.Management* namespace.

5.6.2 Accessing WMI information from Visual Studio.NET

Before doing a brief overview of the .NET coding technique regarding the WMI information access, it is interesting to note that Microsoft provides a WMI-based management extension to the **Visual Studio.NET Server Explorer** tool. This component adds two new nodes to the **Visual Studio.NET Server Explorer** tool:

- **The Management Data node:** Management Data allows the application developer to browse and modify WMI data as well as invoke methods.
- **The Management Events node:** Management Events enable the application developer to register for WMI events.

This add-on can be downloaded from http://www.microsoft.com/downloads/release.asp?ReleaseID=31155. Once installed, you can start the **Visual Studio.NET Server Explorer** tool by pressing Ctrl+Alt+S. Then you can browse the "Management Classes" node and, for instance, select the "Disk Volumes" node and expand the tree until you get the drive list shown in Figure 5.35.

By selecting the C: drive, you will recognize the information exposed by the *Win32_LogicalDisk* class (display name "Disk Volumes" for the C: instance). Note that the **Server Explorer** tool doesn't show the real class name; rather, it shows the class display name. The class display name corre-

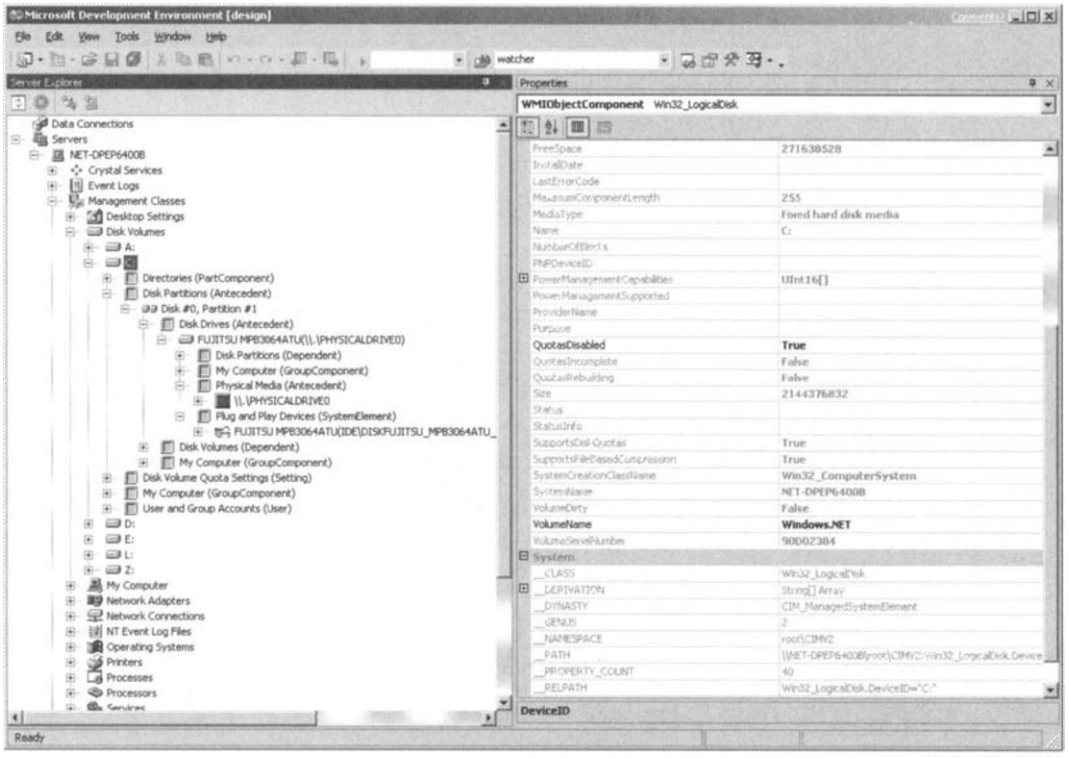

Figure 5.35 *The WMI-based management extension to the Visual Studio.NET Server Explorer tool.*

sponds to the value stored in the **displayname** Qualifier available from the WMI class definition. This could be confusing the first time, since you likely worked with real class names rather than class display names (especially in the WMI COM environment). As shown in Figure 5.35, the *Win32_DiskVolume* class has a "Disk Volumes" display name, where "Disk Volumes" is the value assigned to the **displayname** Qualifier of the *Win32_DiskVolume* class.

By expanding the "Disk Volumes," the **Server Explorer** shows all existing instances of the class (i.e., A:, C:, D:, and E:). Next, if you expand one instance of the *Win32_LogicalDisk* class (i.e., C:), the **Server Explorer** will show all class display names of instances associated with the expanded instance (i.e., directories, partitions, volume quota, etc.). Behind the scenes, **Server Explorer** uses the association classes defined in the CIM repository to establish the link. Moreover, **Server Explorer** allows you to see the properties of a class or instance and execute methods from classes (static methods) or instances (nonstatic methods). If you plan to execute a method, the **Visual Studio.NET Server Explorer** tool will prompt you for the method input parameters. Once the method executes, it will show you the method execution result with the eventual output parameters. However, by default, not all classes from the CIM repository are visible from the **Visual Studio.NET Server Explorer** tool. By right-clicking on the Management Data node and selecting Add Class from the context menu, you get the dialog shown in Figure 5.36.

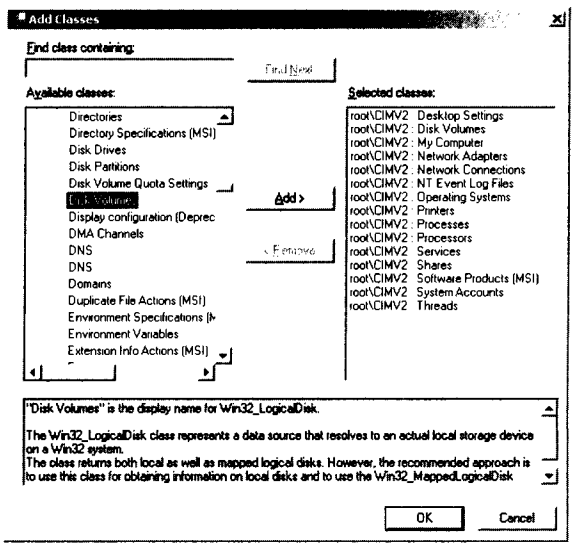

Figure 5.36 *Browsing WMI namespaces for classes.*

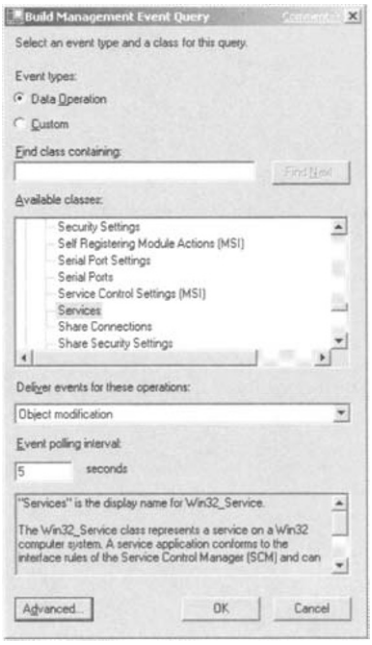

Figure 5.37 *Event subscription from Visual Studio.NET.*

This window allows you to search and browse in the CIM repository across all namespaces and class names. Again, this window shows the class display names, but note that the real class name is visible in the description at the bottom of the window.

With this extension, it is also possible to subscribe to WMI events. For this, you need to right-click on the "Management Events" node and select the "Add Event Query" selection. At that time, the window shown in Figure 5.37 appears.

First, you must select a class from the "Available Classes" window. In Figure 5.37, the "Services" display name corresponds to the *Win32_Service* class. Once the class display name is selected and the condition of event delivery selected with an eventual polling interval, the event query configured in Figure 5.37 will correspond to the following WQL event query:

```
Select * From __InstanceModificationEvent Within 5 Where TargetInstance ISA 'Win32_Service'
```

Once a modification occurs to one of the Windows services, **Visual Studio.NET** will show the instance modification, as shown in Figure 5.38 in the "Output" window (bottom of the screen).

To summarize, we can say that the WMI-based management extension to the **Visual Studio.NET Server Explorer** tool is an updated, integrated,

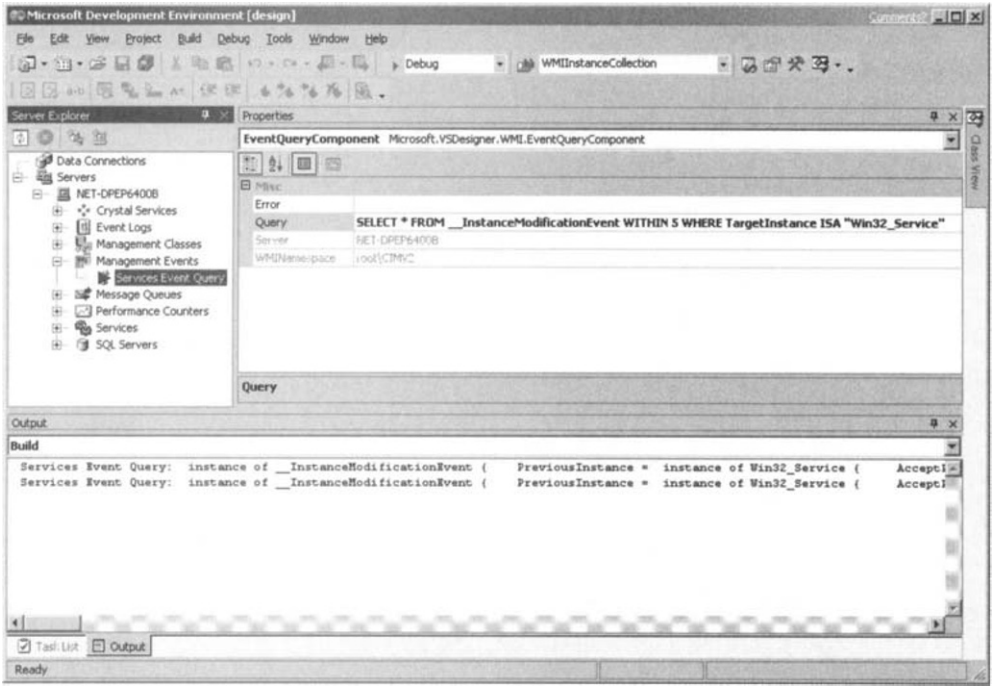

Figure 5.38 *Viewing events from Visual Studio.NET.*

and combined version of **WMI CIM Studio**, **WMI Object Browser**, and **WBEMTEST.Exe** tools.

5.6.3 Accessing WMI information from a .NET language

Even if the object model is different from the one we used in the scripting environment, the type of functionality provided by the .NET *System.Management* classes is similar to the one provided by the WMI COM interfaces. All properties and methods exposed by the WMI COM interfaces have their equivalents in the *System.Management* classes. As before, the event subscriptions and method executions are available as well. Of course, because the .NET Framework abstracts WMI COM interfaces with the *System.Management* and the *System.Management.Instrumentation* classes, it is clear that an adapted coding technique from a language such as C# or VB.NET is required. We won't cover all aspects and details of .NET programming in C# or VB.NET here. Since .NET is an entirely new implementation in several strategic areas, I encourage you to gather more

information about this subject, which is mainly available from the Microsoft .NET Framework SDK. Today, there are loads of publications available on this matter as well. However, through some small examples, we will quickly see that the functionality level you get from the .NET *System.Management* classes is similar to the one you have from the WMI COM API. If you master the capabilities of the WMI COM API (and you should, as you arrive at the end of this book), it will be quite easy to reproduce logics developed in VBScript or Jscript from WSH with a .NET language and the *System.Management* classes.

Because the startup is always the most painful step, let's examine a very basic WMI sample in C#, performing a WMI connection and listing all properties of all instances of the *Win32_Service* class (see Sample 5.23).

Sample 5.23 *Connecting and retrieving all Windows Services instances with their properties with the System.Management classes with C#*

```
using System;
using System.Management;

class Constants
            {
            public const string UserID = "LISSWARENET\\Administrator";
            public const string Password = "password";
            public const string ComputerName = "net-dpen6400a.LissWare.NET";
            public const string NameSpace = "Root\\CIMv2";
            public const string WMIClass = "Win32_Service";
            }

class WMISample
      {
      public static int Main(string[] args)
            {
            ConnectionOptions options = new ConnectionOptions();
            options.Username = Constants.UserID;
            options.Password = Constants.Password;
            options.Authentication = AuthenticationLevel.Default;
            options.Impersonation = ImpersonationLevel.Impersonate;

            ManagementScope scope = new ManagementScope();
            scope.Path = new ManagementPath("\\\\" + Constants.ComputerName +
                                            "\\" + Constants.NameSpace);
            scope.Options = options;

            try {
                scope.Connect();
                }
            catch (Exception Err)
                {
                Console.WriteLine("Failed to connect: " + Err.Message);
                return 1;
                }

```

```
37:             ManagementClass WMIClass = new ManagementClass(Constants.WMIClass);
38:             WMIClass.Scope = scope;
39:
40:             ManagementObjectCollection WMIInstanceCollection = WMIClass.GetInstances();
41:
42:             foreach (ManagementObject WMIInstance in WMIInstanceCollection)
43:                     {
44:                     Console.WriteLine();
45:                     PropertyDataCollection.PropertyDataEnumerator propertyEnumerator =
46:                                             WMIInstance.Properties.GetEnumerator();
47:                     while (propertyEnumerator.MoveNext())
48:                           {
49:                           PropertyData property = (PropertyData)propertyEnumerator.Current;
50:                           Console.WriteLine(property.Name + " = " + property.Value);
51:                           }
52:                     }
53:             return 0;
54:             }
55:     }
```

Sample 5.23 executes the following WMI operations:

- **Performs a WMI remote connection** (lines 17 through 35): The remote connection parameters are stored in an object called **options** made from the *ConnectionOptions* .NET class (line 17). Each parameter of this object holds one parameter for the connection. For instance, the *username* and *password* properties contain the credentials (lines 18 and 19), while the *authentication* and *impersonation* properties determine the WMI authentication and impersonation mode, respectively (lines 20 and 21). Next, the system name and the WMI namespace to access are specified in a **ManagementScope** object created from the *ManagementScope* class (lines 23 through 25). The **ManagementScope** object information is completed with the connection parameters previously created (line 26). Once done, the WMI connection is executed (line 29). In case of error, the exception is handled with a "Try ... catch" C# statement (lines 28 through 35). All connection parameters are defined in constants. These values are available from the *Constants* .NET class defined from line 4 through 11.
- **Retrieves the list of Windows Services available** (lines 37 and 40): Before retrieving all Windows Services available, the code sample creates a **ManagementClass** object from the *ManagementClass* .NET class (line 37). This object represents the WMI class given in reference during the object creation, which is the *Win32_Service* WMI class defined as a constant at line 10. With the **ManagementClass** object, the C# code requests all instances of the *Win32_Service* WMI class by invoking the *GetInstance()* .NET method (line 40). Note that

the **ManagementClass** object knows about the connection details, since it reuses the **ManagementScope** object created for the connection (line 38). Because the connection is previously established (line 29) and because the *ManagementClass* object reuses the same **ManagementScope** object, the connection is not recreated and the connection channel is reused. The *GetInstance()* .NET method invocation creates a new **ManagementObjectCollection** object (line 40), which contains a collection of all Windows Services instances available from the connected system.

- **For each Windows service in the collection, the code retrieves all properties with their values** (lines 42 through 52): To display the Windows services available, the **ManagementObjectCollection** object must be enumerated. Therefore, a loop performs this enumeration (lines 42 through 52). For each Windows service instance we have in the collection, the C# code retrieves the property list available from the instance by creating a **PropertyDataCollection.PropertyDataEnumerator** object (lines 45 and 46). Next, for each property we have in this new collection, the code enumerates each property one by one (lines 47 through 51) and retrieves the property in a **PropertyData** object (line 49) to display its name and its corresponding value (line 50).

As we can see in Sample 5.23, we perform a task we already did many times from WSH. From a functional point of view, there is no difference. However, the coding and the object references are totally different. This makes sense, since we use another language (i.e., C#) and programming object model.

We can also submit WQL queries as we did from the WMI COM interfaces. Sample 5.24 uses the same structure to retrieve the same set of information as Sample 5.23. However, instead of invoking the *GetInstances()* .NET method, it performs a WQL data query.

Sample 5.24 *Connecting and retrieving all Windows Services instances with their properties with a WQL data query with the System.Management classes with C#*

```
1:  using System;
2:  using System.Management;
3:
4:  class Constants
5:        {
6:        public const string UserID = "LISSWARENET\\Administrator";
7:        public const string Password = "password";
8:        public const string ComputerName = "net-dpen6400a.LissWare.NET";
9:        public const string NameSpace = "Root\\CIMv2";
```

```
10:         public const string WQLQuery = "Select * From Win32_Service";
11:         }
12:
13:   class WMISample
14:         {
15:         public static int Main(string[] args)
16:                {
..:
37:                ManagementObjectSearcher WMIInstanceSearcher = new ManagementObjectSearcher();
38:                WMIInstanceSearcher.Query = new ObjectQuery(Constants.WQLQuery);
39:                WMIInstanceSearcher.Scope = scope;
40:
41:                ManagementObjectCollection WMIInstances = WMIInstanceSearcher.Get();
42:
43:                foreach (ManagementObject WMIInstance in WMIInstances)
44:                       {
..:
53:                       }
54:                return 0;
55:                }
56:          }
```

The first modification concerns line 10. This line defines the WQL data query in a new constant. On line 38, this constant is referenced to create an **ObjectQuery** object made from the *ObjectQuery* .NET class. Next, this object is stored in a new **ManagementObjectSearcher** object (line 38) created previously (line 37) from the *ManagementObjectSearcher* .NET class. As in Sample 5.23, the **ManagementObjectSearcher** object reuses the **ManagementScope** object created previously (skipped lines 16 through 37) to establish the WMI connection (line 39). Next, a **ManagementObjectCollection** object is created (line 41). Instead of creating this object from the *GetInstance()* .NET method, it is created by invoking the WMI search with the *Get()* .NET method exposed by the **ManagementObjectSearcher** object. The search is based on the WQL data query assigned to the **ManagementObjectSearcher** object (line 38). Next, as before, the C# code enumerates all instances available with their properties and values (lines 43 through 53).

If we talk about WQL data queries, we should talk about WQL event queries as well. This is the purpose of Sample 5.25, which performs an asynchronous WQL event query.

Sample 5.25 *Connecting and performing a WQL event query to display Windows Services instances subject to a modification with the System.Management classes with C#*

```
1:   using System;
2:   using System.Management;
3:
4:   class Constants
5:              {
6:              public const string UserID = "LISSWARENET\\Administrator";
```

```
 7:              public const string Password = "password";
 8:              public const string ComputerName = "net-dpen6400a.LissWare.Net";
 9:              public const string NameSpace = "Root\\CIMv2";
10:              public const string WQLQuery = "SELECT * FROM __InstanceModificationEvent" +
11:                                             "WITHIN 5 " +
12:                                             "WHERE TargetInstance ISA \"Win32_Service\"";
13:              }
14:
15:    class WMISample
16:          {
17:          public static int Main(string[] args)
18:                 {
...:
30:                 WQLEventQuery WQLEventQuery = new WqlEventQuery();
31:                 WQLEventQuery.QueryString = Constants.WQLQuery;
32:
33:                 ManagementEventWatcher WMIEventWatcher = new ManagementEventWatcher();
34:                 WMIEventWatcher.Scope = scope;
35:                 WMIEventWatcher.Query = WQLEventQuery;
36:
37:                 WMIEventHandler WMIHandler = new WMIEventHandler();
38:                 WMIEventWatcher.EventArrived +=
39:                                 new EventArrivedEventHandler(WMIHandler.Arrived);
40:
41:                 try {
42:                     WMIEventWatcher.Start();
43:                     }
44:                 catch (Exception Err)
45:                     {
46:                     Console.WriteLine("Failed to connect: " + Err.Message);
47:                     return 1;
48:                     }
49:
50:                 Console.WriteLine("Waiting for events ...");
51:
52:                 while (!WMIHandler.IsArrived)
53:                       {
54:                       System.Threading.Thread.Sleep(1000);
55:                       }
56:
57:                 WMIEventWatcher.Stop();
58:
59:                 return 0;
60:                 }
61:          }
62:
63:          public class WMIEventHandler
64:                 {
65:                 private bool isArrived = false;
66:
67:                 public void Arrived(object sender, EventArrivedEventArgs EventArgs)
68:                        {
69:                        ManagementBaseObject WMIInstance =
70:                                  (ManagementBaseObject)(EventArgs.NewEvent["TargetInstance"]);
71:
72:                        Console.WriteLine("Event triggered ...\n" );
73:                        PropertyDataCollection.PropertyDataEnumerator propertyEnumerator =
74:                                                WMIInstance.Properties.GetEnumerator();
75:                        while (propertyEnumerator.MoveNext())
76:                              {
```

```
77:                         PropertyData property = (PropertyData)propertyEnumerator.Current;
78:                         Console.WriteLine(property.Name + " = " + property.Value);
79:                         }
80:
81:                         isArrived = true;
82:                     }
83:
84:                 public bool IsArrived
85:                     {
86:                     get
87:                       {
88:                       return isArrived;
89:                       }
90:                     }
91:                 }
```

This last .NET C# example is slightly more complex. As with previous examples, Sample 5.25 starts by creating the **ConnectionOptions** and **ManagementScope** objects (lines 19 through 28). Next, it creates the **WQLEventQuery** object holding the WQL event query (lines 30 and 31) defined as a constant in the *Constants* class (lines 10 through 12). To briefly demonstrate the versatility of the *System.Management* classes, the WQL query can be assigned to the **WQLEventQuery** object in many different ways. In Sample 5.24, we used one coding technique (line 38). In Sample 5.25, we use the following assignment, where we split the object creation from the WQL query assignment:

```
WQLEventQuery WQLEventQuery = new WqlEventQuery();
WQLEventQuery.QueryString = Constants.WQLQuery;
```

However, in the *System.Management* namespace object model, the WQL query can also be specified parameter by parameter:

```
WQLEventQuery WQLEventQuery = new WqlEventQuery("__InstanceModificationEvent", new TimeSpan(0,0,5),
                                                "TargetInstance ISA 'Win32_Service'");
```

Or, eventually, the WQL query parameters can be assigned to their respective properties one by one:

```
WQLEventQuery WQLEventQuery = new WqlEventQuery();
WQLEventQuery.EventClassName = "__InstanceModificationEvent";
WQLEventQuery.Condition = "TargetInstance ISA 'Win32_Service'";
WQLEventQuery.WithinInterval = new TimeSpan(0,0,5);
```

All these WQL query assignments produce the exact same result: They create a **WQLEventQuery** object made from the *WQLEventQuery* .NET class containing the WQL query defined in the *Contants* .NET class. This small example shows how versatile the object model can be and how it can be confusing for beginners as well!

Next, the portion of code preparing the ground for the asynchronous event monitoring starts with the creation of a **ManagementEventWatcher**

object (lines 33 through 39). This object contains the connection settings (line 34) and the **WQLEventQuery** object (line 35). Since the monitoring is asynchronous, a **WMIEventHandler** object is created (line 37) from the *WMIEventHandler* .NET class defined further in the code (lines 63 through 91). This class implements the event sink routine handling WMI event notifications. We will come back to this class creation later in this section.

Once the **ManagementEventWatcher** object is created (line 33), in addition to the **ManagementScope** and **WQLEventQuery** objects information, the **ManagementEventWatcher** object needs to know which routine is used as the event handler. Therefore, based on the event type, the **ManagementEventWatcher** object is initialized with a pointer to the event routine. The event types can be:

- **EventArrived:** Occurs when a new event arrives.
- **Stopped:** Occurs when a subscription to an event is canceled.

```
38:             WMIEventWatcher.EventArrived +=
39:                             new EventArrivedEventHandler(WMIHandler.Arrived);
```

For people not used to the C# notation, the VB.NET notation may make more sense:

```
AddHandler WMIEventWatcher.EventArrived, AddressOf WMIHandler.Arrived
```

Next, the event watching is started (line 42) by invoking the *Start()* method of the **ManagementEventWatcher** object. Since the WMI connection is performed during the event watching startup process, the statement is enclosed in a "Try ... catch" C# statement to manage exceptions (lines 41 through 48). Next, the C# code will wait until an event arrives (lines 52 through 55). The *IsArrived* property is exposed by the *WMIEventHandler* .NET class created within the C# code (lines 84 through 90).

The last piece of code to examine concerns the event handler itself (lines 63 through 91). The event handler is triggered for the **EventArrived** event. Therefore, a subfunction, which must be named *Arrived()*, is created inside the .NET class (lines 67 to 82). Two .NET objects are passed as parameters along the event notification:

- **Sender** object: This object contains information about the event sender. For instance, with this object it is possible to retrieve information contained in the **WMIEventWatcher** object created in the first part of the code. This could be done as follows inside the event handler routine:

```
ManagementEventWatcher WMIEventWatcher = (ManagementEventWatcher) sender;
Console.WriteLine(WMIEventWatcher.Query.QueryString);
```

- **EventArgs** object: This object contains information about the event itself and is made from the *EventArrivedEventArgs* .NET class. This object exposes two properties: the *Context* property and the *NewEvent* property. The *NewEvent* property gets information about the WMI event notification. Therefore, to retrieve information about the WMI instance subject to the *__InstanceModificationEvent*, the C# code retrieves, via the WMI *TargetInstance* property exposed by the WMI instance event notification, the *Win32_Service* instance subject to the changes (line 69 and 70). This instance is stored in a **ManagementBaseObject** object made from the *ManagementBaseObject* .NET class.

```
68:         ManagementBaseObject WMIInstance =
69:                 (ManagementBaseObject)(EventArgs.NewEvent["TargetInstance"]);
```

Once the **ManagementBaseObject** object is available, Sample 5.25 proceeds as previous samples: It enumerates all properties with their values of the WMI instance (lines 72 through 82). Once the enumeration is completed, the *IsArrived* property exposed by the *WMIEventHandler* .NET class is updated (line 81), which makes the main routine of Sample 5.25 exiting from its loop at lines 52 through 55. Then the event watcher is stopped (line 57) before the C# code terminates (line 59).

5.7 Summary

Today, most applications released by Microsoft are WMI enabled. Even if we only reviewed some of the most important applications produced by Microsoft, there are many other applications supporting WMI that we didn't talk about, such as Host Integration Server (Microsoft SNA Server), System Management Server (SMS), or Services For Unix 3.0 for some existing products. At writing time, some new products, still under development, will also make an extensive usage of a WMI. To give some examples:

- Exchange Server 2003 extends the current Exchange 2000 WMI capabilities for administration purposes.
- Automated Deployment Service provides a new set of WMI classes for its administration and automation purposes.
- Windows System Resource Manager makes use of the SMTP event consumer for alerting purposes.
- Microsoft Meta-Directory Service 3.0 provides a new set of WMI classes for its administration and its internal instrumentation.

To give another example in the area of the Enterprise management software, HP OpenView Operations for Windows or Microsoft Operation Manager utilize WMI information to manage Windows at the enterprise level while providing WMI information at the same time for manageability and customization. Today, it is inconceivable that management applications in the Windows space are not WMI enabled. WMI is the foundation of a solid enterprise management solution for everyone in charge of managing or administering Windows platforms today.

5.8 Useful Internet URLs

Management (WMI) Extensions for Visual Studio .NET RTM Server Explorer

```
http://www.microsoft.com/downloads/release.asp?ReleaseID=31155
```