1

The Windows WMI Providers Discovery

1.1 Objective

During the initial WMI discovery to illustrate the scripting technique, we used a very few number of classes from the CIM repository (see the first book about WMI, *Understanding WMI Scripting*). Mostly, we worked with the *Win32_Service* class, *Win32_WMISetting,* and *Win32_NTLogEvent* classes. We also examined the WMI event model and worked with some specific event classes. The standard WMI installation includes more than 600 classes that expose more than 3,000 properties. As we can see, we are a long way from completing our WMI discovery! Classes representing manageable entities of the real world have a direct relation with the WMI providers registered in the CIM repository. Although the Microsoft Platform SDK documents most of the WMI providers available, we discuss here how to retrieve information about the WMI providers and the class they support by extracting information from the CIM repository. This information will serve as the basis for subsequent chapters when discovering the most important WMI providers available with their classes and capabilities.

1.2 The WMI provider types

A WMI provider registered in the CIM repository is an instance of the *__Win32Provider* system class. Basically, the *__Win32Provider* system class registers information about a provider's physical implementation with WMI. So, requesting all instances of the *__Win32Provider* system class in a particular WMI namespace will list all registered providers in that particular namespace.

During the WMI discovery, we saw that a set of system classes is available to define the nature of the providers (see Figure 1.1).

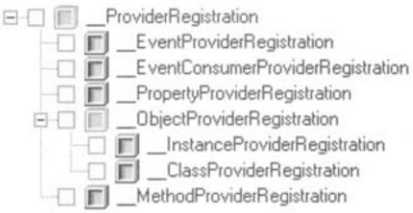

Figure 1.1 *The provider registration system classes.*

WMI defines a certain number of provider types, which determine the nature of the information delivered by the providers. These providers are as follows:

- **The class providers:** These providers supply applications with class definitions. They are rarely implemented, because classes are usually stored in the CIM repository. In most cases, the classes are slow-changing and finite. If classes need to be dynamically generated, then a class provider must be implemented, but this slows down the performance of information retrieval. This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__ClassProviderRegistration* system class (see Figure 1.2). The *Active Directory*

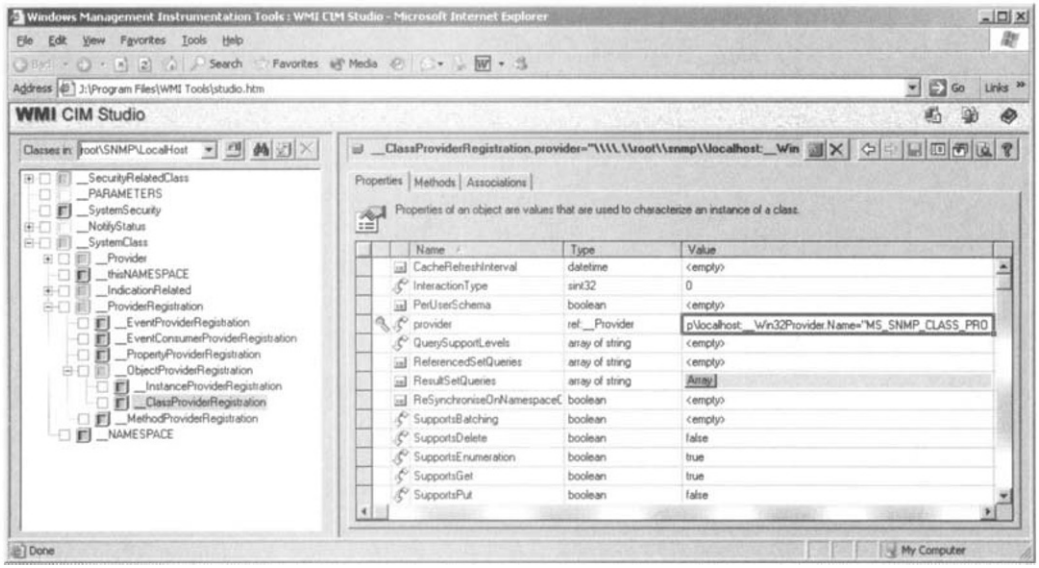

Figure 1.2 *The SNMP class provider registration.*

providers and the *SNMP* providers are two examples of providers that implement a class provider.

- **The instance providers:** These providers are the most common providers and supply instances of a given class. Usually, they provide services such as: instance retrieval, enumeration, modification, deletion, and query processing. This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__InstanceProviderRegistration* system class (see Figure 1.3). For instance, the *Win32* provider and the *Registry* provider are implemented as instance providers.

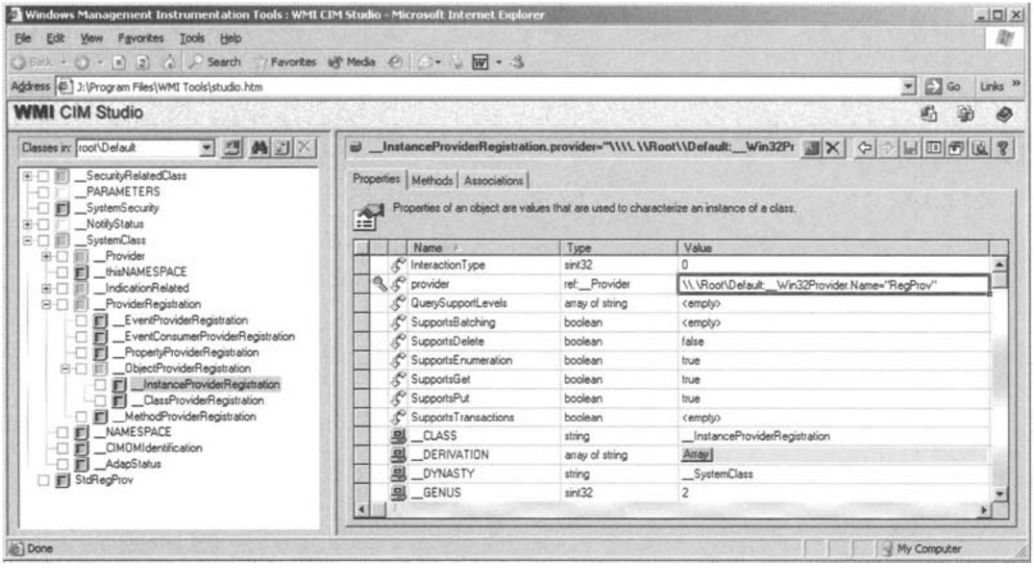

Figure 1.3 *The Registry instance provider registration.*

- **The property providers:** These providers can support the retrieval and the modification of individual property values. This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__PropertyProviderRegistration* system class (see Figure 1.4). The *Registry* provider is also implemented as a property provider.

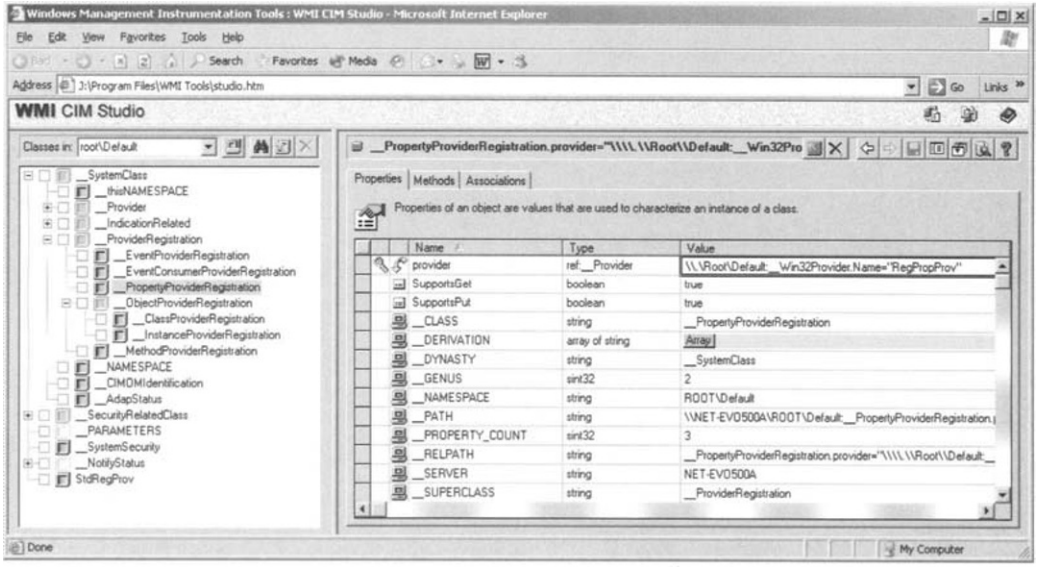

Figure 1.4 *The Registry property provider registration.*

- **The methods providers:** These providers implement the methods of one or more classes. This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__MethodProviderRegistration* system class (see Figure 1.5). For example, the *Win32* provider and the *Registry* provider are implemented as method providers as well.
- **The event providers:** These providers deliver event notifications to WMI. Next, WMI forwards the events to the appropriate event consumers (temporary or permanent, based on the consumer type). This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__EventProviderRegistration* system class (see Figure 1.6). For instance, the *SNMP* provider, the *NT Event Log* provider, and the *Registry* provider are implemented as event providers.
- **The event consumer providers:** These providers were examined during the WMI discovery. They simply act as consumers of WMI events. This type of provider is registered in the CIM repository with an instance of the *__Win32Provider* system class, which has a reference to an instance of the *__EventConsumerProviderRegistration* system class.

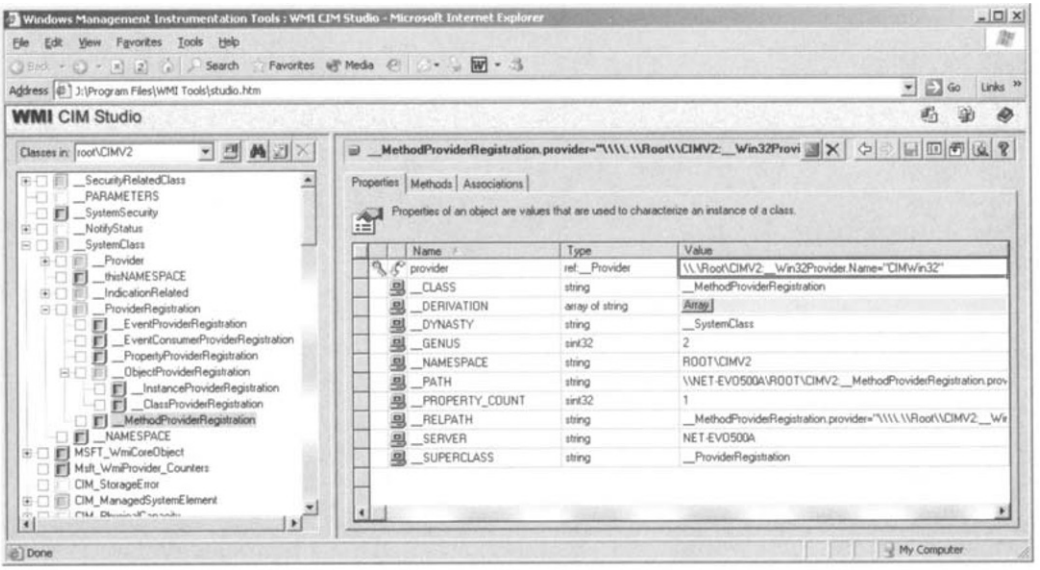

Figure 1.5 *The Win32 method provider registration.*

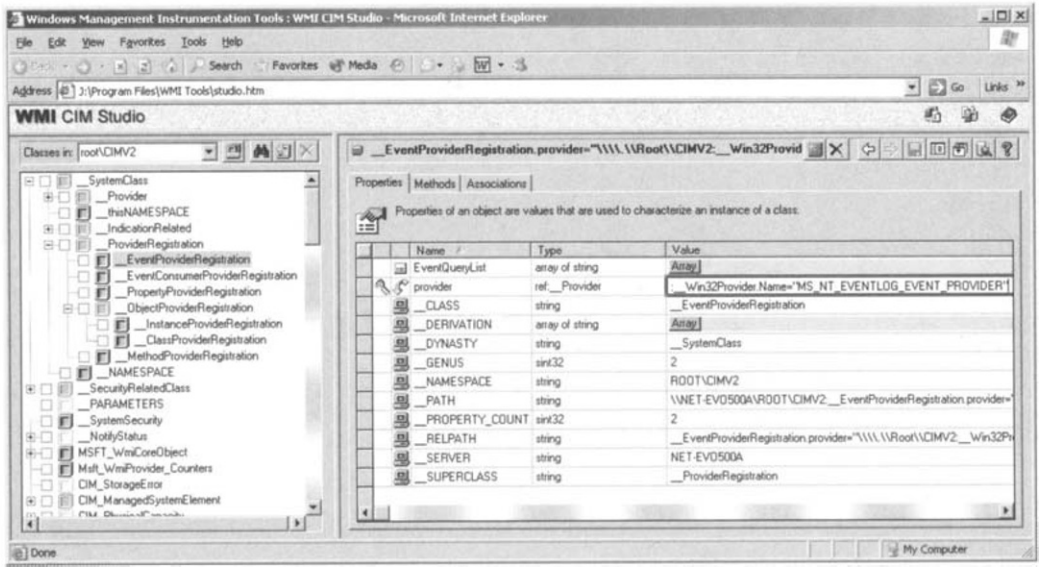

Figure 1.6 *The NT Event Log event provider registration.*

The *Command-Line* event consumer, the *SMTP* event consumer, and the *Active Script* event consumer are examples of event consumer providers (see Chapter 6 in *Understanding WMI Scripting*).

Among these provider types, WMI also supports high-performance providers. These providers greatly increase the speed at which WMI clients can obtain information from WMI. They are nothing more than instance providers implemented as Dynamic-Link Libraries (DLL). We will revisit this provider type later in this book when examining the *Performance Counter* provider and the *Cooked Counter* provider in Chapter 3, section 3.8.

Throughout *Understanding WMI Scripting*, we explain the difference between a class and an instance. While examining the properties exposed by some classes representing real-world manageable entities, we have seen that some classes only expose properties (i.e., *Win32_WMISetting* or *Win32_NTLogEvent* classes), while others expose both properties and methods (i.e., *Win32_Service* class). Because the characteristics implemented in the classes are nothing more than an object model representing the real-world manageable entities, it is clear that the primary role of the WMI provider is to provide the real-world information. This implies that a provider will be:

- An instance provider, if it provides instances of real-world manageable entities
- A property provider, if it provides properties
- A method provider, if it provides methods

Since the real-world entities can be instantiated with WMI to represent real-world objects, this also implies that methods can be invoked to perform some actions and that WMI events can be received to monitor modifications. Therefore, it is very common that a single provider combines different roles at the same time. For instance, the *Registry* provider is made of one single DLL called **STDPROV.DLL**, which exposes three COM objects. These three COM objects implement the four provider types used by WMI to manage the registry:

- The first COM object implements an instance and a method provider. It is called *RegProv*.
- The second COM object implements a property provider and is called *RegPropProv*.
- The third COM object implements an event provider and is called *RegistryEventProvider*.

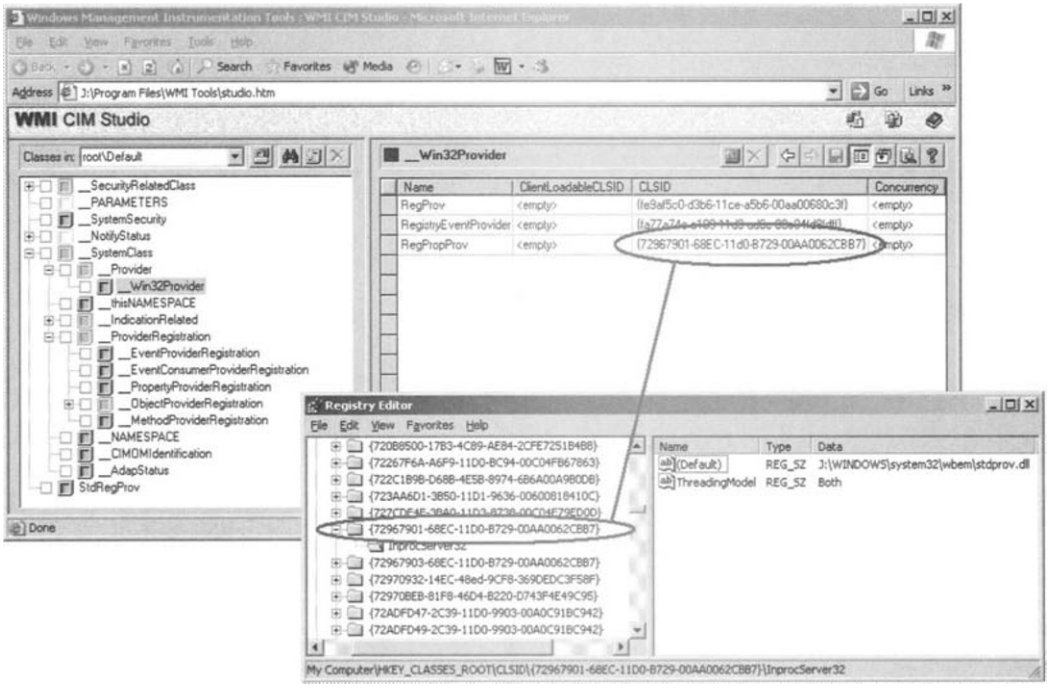

Figure 1.7 *The Registry provider registration in the CIM repository and in the registry.*

Therefore, the *Registry* provider must be registered accordingly in the CIM repository with:

- Three instances of the *__Win32Provider* system class for its physical implementation (**STDPROV.DLL**), where each instance contains the CLSID of the COM object registered in the registry (see Figure 1.7).
- An instance of the *__InstanceProviderRegistration* system class with a reference to its *__Win32Provider* instance.
- An instance of the *__MethodProviderRegistration* system class with a reference to its *__Win32Provider* instance.
- An instance of the *__EventProviderRegistration* system class with a reference to its *__Win32Provider* instance.

When we began the WMI discovery, we saw what a reference and an association are (please refer to Chapters 2, 3, and 4 in *Understanding WMI Scripting* for more information on this subject). We learned how to locate references and associations that exist with a class and an instance. This knowledge is useful, because this is the way to determine the provider type.

Each time an instance of the *__Win32Provider* system class is available, we have an instance of a WMI provider. By examining the references available from that instance, we determine the WMI provider type (Instance provider, Method provider, etc.). Doing so, we have created a WMI provider discovery technique that helps us to understand some of the WMI provider capabilities, which also determine the behavior of the WMI classes they support.

1.3 WMI providers discovery

Some of you may wonder why we need to discover the WMI provider existence with their classes when they are documented in the WMI SDK. First, not all providers available from WMI are documented in the WMI SDK, which implies that not all available WMI classes are documented. Mastering a discovery technique will allow you to understand how the CIM repository defines the existence of its providers and which class they support. The discovery will help to acquire knowledge about the WMI provider capabilities and get a better understanding about what it is possible to do when using their classes. This is why it is useful to have a WMI provider discovery technique. Moreover, if tomorrow a new application registers some new providers and adds new classes in the CIM repository, mastering the discovery technique immediately gives you the required information to develop an application or a script that leverages these new classes, even if they are not documented by the application developer (i.e., because the classes are intended for the application's internal use only).

As we have seen in the previous section, we know that every WMI provider registered in the CIM repository creates some instances from a set of system classes and that the set of system classes used depends on the WMI provider type. Sample 4.30 in the appendix shows how a script can browse the CIM repository to locate WMI instances across the namespaces. If we reuse that piece of code to locate the instances of the *__Win32Provider* system class used to register WMI providers across all existing namespaces, we will locate all registered WMI providers in the CIM repository. To be complete and determine the WMI provider type (Instance provider, Method provider, etc.), we must also check if each *__Win32Provider* instance has some associated references.

Listing the WMI providers with their types is only a piece of the desired information, because we want to know what classes are supported by the

providers. This information is available in the class definition itself. The WMI provider that a class relates to depends on the provider type:

- For the class providers, the Instance providers, the Method providers, and the Property providers, the *provider* qualifier is set in the class definition, which determines the provider supporting the class (see Figure 1.8).

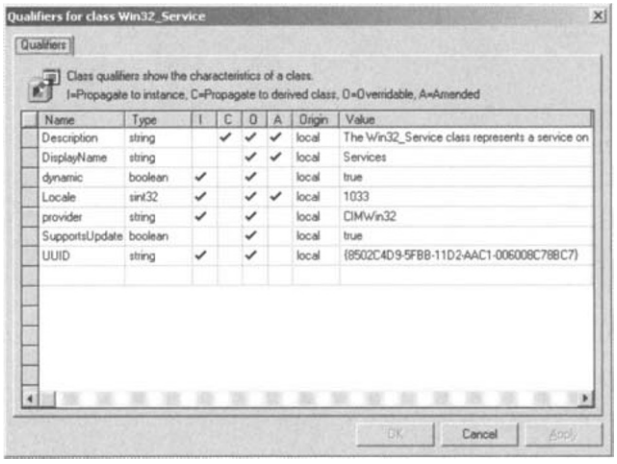

Figure 1.8 *The provider qualifier of a class to determine the supported WMI provider.*

- For the event providers, the *__EventProviderRegistration* system class exposes a property called *EventQueryList*. This property contains a collection of the WQL queries supported by the event provider. In these WQL queries, the various classes supported by the event provider are mentioned (see Figure 1.9). These classes are categorized as intrinsic and extrinsic event classes.
- For the event consumer providers, the *__EventConsumerProviderRegistration* system class exposes a property called *ConsumerClassNames*. This property contains a collection of classes supported by the event consumer provider (see Figure 1.10).

In *Understanding WMI Scripting*, Chapter 4, we saw how to retrieve the qualifiers of a class and enumerate a collection of properties. We can reuse this knowledge to develop script logic that retrieves the registered providers with the classes they provide. This logic is implemented in Samples 1.1 through 1.3.

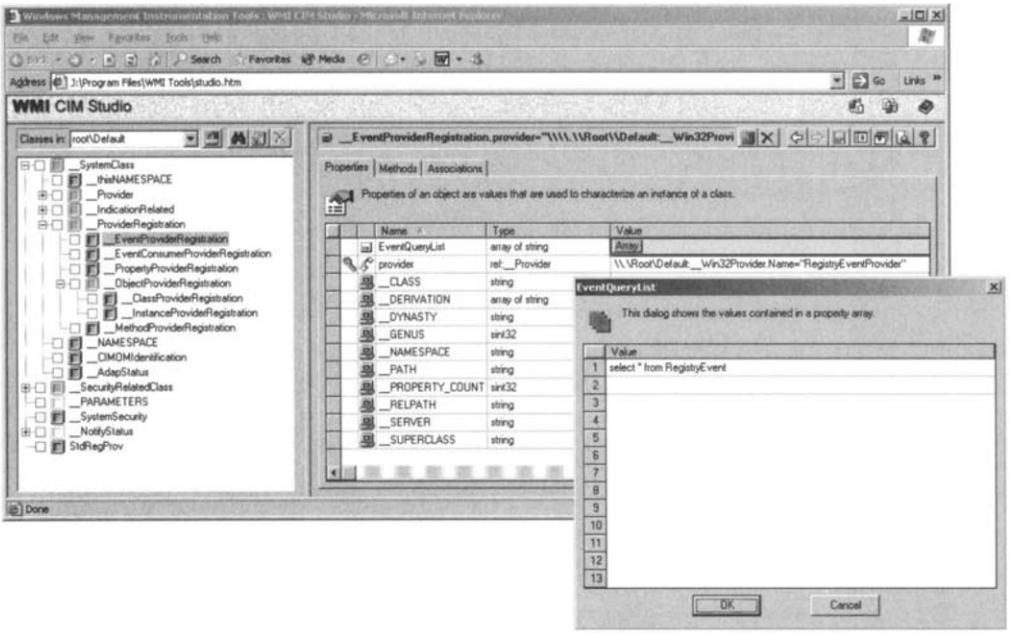

Figure 1.9 *The EventQueryList property of one __EventProviderRegistration instance.*

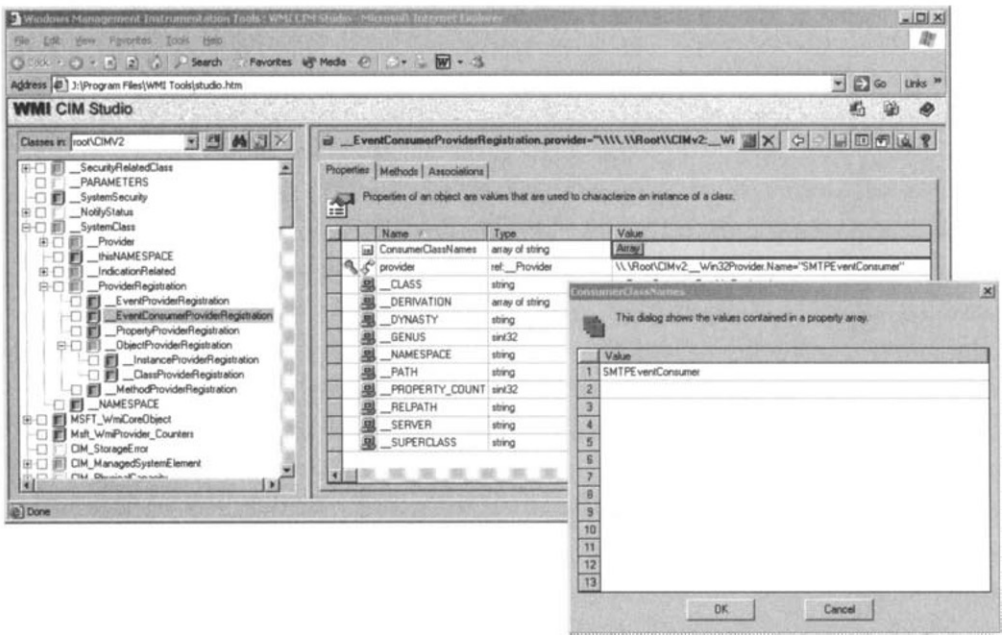

Figure 1.10 *The ConsumerClassNames property of one __EventConsumerProviderRegistration.*

Sample 1.1 *Locating WMI provider registration instances with their supported classes (Part I)*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9: <job>
..:
13:  <runtime>
14:    <named name="Machine" helpstring="determine the WMI system to connect to.
                (default=LocalHost)" required="false" type="string"/>
15:    <named name="User" helpstring="determine the UserID to perform the remote connection.
                (default=none)" required="false" type="string"/>
16:    <named name="Password" helpstring="determine the password to perform the remote connection.
                (default=none)" required="false" type="string"/>
17:  </runtime>
18:
19:  <script language="VBScript" src=".\Functions\DisplayInstanceProperties.vbs" />
20:  <script language="VBScript" src=".\Functions\TinyErrorHandler.vbs" />
21:
22:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
23:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
24:
25:  <script language="VBscript">
26:  <![CDATA[
..:
30:  Const cComputerName = "LocalHost"
31:  Const cWMINameSpace = "Root"
32:  Const cWMINamespaceclass = "__NAMESPACE"
33:  Const cWMIclass = "__Win32provider"
..:
39:  ' ------------------------------------------------------------------------------
40:  ' Parse the command line parameters
41:  strUserID = WScript.Arguments.Named("User")
42:  If Len(strUserID) = 0 Then strUserID = ""
43:
44:  strPassword = WScript.Arguments.Named("Password")
45:  If Len(strPassword) = 0 Then strPassword = ""
46:
47:  strComputerName = WScript.Arguments.Named("Machine")
48:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
49:
50:  DisplayNameSpaces cWMINameSpace, cWMIclass, strUserID, strPassword, strComputerName
51:
..:
..:
..:
```

Sample 1.1 contains the first part of the script. This part uses a structure that we have already seen many times in the first book, *Understanding WMI Scripting*. Lines 13 through 17 contain the parameter definitions that can be used from the command line. Next, the script processes the command-line parameters analysis (lines 39 through 48).

Once the command-line parameters analysis completes, the script calls the DisplayNameSpaces() function (line 50). Note that the parameters

passed to the function are the **Root** namespace (defined in line 31) and the *__Win32Provider* system class (defined in line 33). The DisplayNameSpaces() function uses the same parameters as Sample 4.30 (see the appendix) to browse the namespaces defined in the CIM repository. Although the function is still using a recursive model, the internal logic has been slightly modified to suit our needs. Sample 1.2 shows the updated version of this function.

Sample 1.2 *Locating WMI provider registration instances with their supported classes (Part II)*

```
..:
..:
..:
51:
52:  ' ---------------------------------------------------------------------
53:  Private Function DisplayNameSpaces (ByVal strWMINameSpace, _
54:                                      ByVal strWMIclass, _
55:                                      ByVal strUserID, _
56:                                      ByVal strPassword, _
57:                                      ByVal strComputerName)
..:
66:    objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
67:    objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
68:    Set objWMIServices = objWMILocator.ConnectServer(strComputerName, _
69:                                strWMINameSpace, _
70:                                strUserID, _
71:                                strPassword)
72:    If Err.Number Then ErrorHandler (Err)
73:
74:    Set objWMIInstances = objWMIServices.ExecQuery ("Select * From " & strWMIclass)
75:    If Err.Number Then
76:       Err.Clear
77:    Else
78:       WScript.Echo strWMINameSpace
79:       For Each objWMIInstance in objWMIInstances
80:
81:         WScript.Echo "  " & String (40, "-")
82:         WScript.Echo objWMIInstance.Name
83:
84:         For Each objWMIReference in objWMIInstance.References_
85:           WScript.Echo objWMIReference.Path_.class
86:           Select Case LCase (objWMIReference.Path_.class)
87:               Case "__eventproviderregistration"
88:                 DisplaySupportedQueryclasses (objWMIReference.EventQueryList)
89:               Case "__eventconsumerproviderregistration"
90:                 DisplaySupportedConsumerclasses (objWMIReference.ConsumerclassNames)
91:           End Select
92:         Next
93:
94:         Displayproviderclasses objWMIServices, "", objWMIInstance.Name
95:       Next
96:       Set objWMIInstances = Nothing
97:    End If
98:
99:    Set objWMINSInstances = objWMIServices.InstancesOf (cWMINamespaceclass, _
100:                               wbemQueryFlagShallow)
```

```
101:    For Each objWMINSInstance in objWMINSInstances
102:        DisplayNameSpaces strWMINameSpace & "/" & objWMINSInstance.Name, _
103:                  strWMIclass, strUserID, strPassword, strComputerName
104:    Next
...:
109:  End Function
110:
...:
...:
...:
```

Once the DisplayNameSpaces() function is called, the code starts by establishing the WMI connection to the examined namespace (lines 66 through 72). Next, the script retrieves all instances of the WMI providers registered in the examined namespace (line 74). For this, it uses the *__Win32Provider* system class passed as a parameter of the DisplayNameSpaces() function (line 50). If no fault occurs, the code enumerates the retrieved collection (lines 79 through 95). The slight change concerns the logic encapsulated in this particular loop. Originally, the DisplayNameSpaces() function displayed the properties of the retrieved instances. Now, it shows the name of the WMI provider instance (line 82) and retrieves the references of the provider instance by using the *References_* property (line 84).

If the references point to an *__EventProviderRegistration* or an *__EventConsumerProviderRegistration* system class, the script locates the supported classes, as previously explained. For this, it calls the DisplaySupportedQueryClasses() (line 87) or DisplaySupportedConsumerClasses() (line 89) functions based on the system class type used by the reference (lines 85 through 91). These two functions simply enumerate the properties content visible in Figures 1.9 and 1.10.

For each reference found, the script shows the class reference (line 85). For instance, if the WMI provider is an instance provider and a method provider, it shows the *__InstanceProviderRegistration* and the *__MethodProviderRegistration* system classes, as shown in the following sample output (lines 9 and 10).

```
 1: C:\>Locateproviders.Wsf
 2: Microsoft (R) Windows Script Host Version 5.6
 3: Copyright (C) Microsoft Corporation 1996-2000. All rights reserved.
 4:
 5: Root
 6: Root/CIMV2
 7: ----------------------------------------
 8: CIMWin32
 9: __MethodproviderRegistration
10: __InstanceproviderRegistration
11: > CIM_LogicalFile
12: > CIM_DataFile
13: > Win32_CodecFile
```

```
14: > Win32_ShortcutFile
15: > Win32_PageFile
16: > Win32_Directory
17: > Win32_ComputerSystem
18: > Win32_DeviceMemoryAddress
19: > Win32_PortResource
20: > Win32_DMAChannel
21: > Win32_IRQResource
22: > Win32_Environment
23: > Win32_LogicalDisk
..:
..:
..:
```

At line 8 of the output, we see the name of the provider. This name is referenced by the *provider* qualifier of the classes supported by the examined WMI provider (shown in Figure 1.8 for the *Win32_Service* class). This element can be used to determine the classes that relate to a WMI provider (lines 11 through 23 and more). This is why, at line 94 of Sample 1.2, the script calls the DisplayProviderClasses() function (see Sample 1.3 for the function code). The function call passes the following parameters:

- The **SWBemServices** object created when establishing the connection to the examined WMI namespace
- The provider name (i.e., *CIMWin32*)
- An empty class name

The empty class name refers to the root class of the examined namespace. Sample 1.3 contains the code that searches the classes supported by the examined WMI provider from the top class hierarchy.

Sample 1.3 *Locating WMI provider registration instances with their supported classes (Part III)*

```
...:
...:
...:
110:
111:  ' ------------------------------------------------------------------------------------
112:  Private Function Displayproviderclasses (ByVal objWMIServices, _
113:                                          ByVal strWMIclass, _
114:                                          ByVal strproviderName)
...:
121:      Set objWMIInstances = objWMIServices.SubclassesOf (strWMIclass, wbemQueryFlagDeep)
122:
123:      For Each objWMIInstance in objWMIInstances
124:          strProvqualifier = objWMIInstance.qualifiers_.item("provider")
125:          If Not Err.Number Then
126:             If Ucase (strproviderName) = Ucase (strProvqualifier) Then
127:                WScript.Echo "> " & objWMIInstance.Path_.RelPath
128:             End If
129:          End If
130:          strProvqualifier = ""
131:      Next
132:
```

```
133:      Set objWMIInstances = Nothing
134:
135:  End Function
136:
137:  ' ------------------------------------------------------------------------------
138:  Private Function DisplaySupportedQueryclasses (ByVal arrayEventQueryList)
...:
144:      For Each strEventQuery in arrayEventQueryList
145:          WScript.Echo "> " & Mid (strEventQuery, _
146:                                   InStrRev (strEventQuery, " ", -1, vbTextCompare) + 1)
147:      Next
148:
149:  End Function
150:
151:  ' ------------------------------------------------------------------------------
152:  Private Function DisplaySupportedConsumerclasses (ByVal arrayConsumerclassList)
...:
158:    For Each strConsumerclass in arrayConsumerclassList
159:      WScript.Echo "> " & strConsumerclass
160:    Next
161:
162:  End Function
163:
164:  ]]>
165:  </script>
166: </job>
167:</package>
```

In the DisplayProviderClasses() function, the code uses the *SubClassesOf* method of the **SWBemServices** object with the **wbemQueryFlagDeep** flag (line 121). This means that the script code retrieves all classes available in the namespace. Don't forget that we use a blank class (line 94) to force a search from the Root of the namespace. Next, the retrieved collection of classes is examined one by one in the **For Each** loop (lines 123 through 131). For each examined class in the collection, the function retrieves the *provider* qualifier (line 124) and compares its content (line 126) with the name of the provider passed during the function call (line 114). If there is a match, it means that the examined provider supports the class. In this case, the class name is displayed (line 127). The output is as follows:

```
..:
..:
..:
11:> CIM_LogicalFile
12:> CIM_DataFile
13:> Win32_CodecFile
14:> Win32_ShortcutFile
15:> Win32_PageFile
16:> Win32_Directory
17:> Win32_ComputerSystem
18:> Win32_DeviceMemoryAddress
19:> Win32_PortResource
20:> Win32_DMAChannel
21:> Win32_IRQResource
22:> Win32_Environment
23:> Win32_LogicalDisk
..:
..:
..:
```

Table 1.1 *The Most Important WMI Providers under the Windows Platforms with Their Capabilities*

Provider Name	Provider Classes	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
ActiveScriptEventConsumer (Optional)	Root/subscription						X					X	X	X		
CommandLineEventConsumer (Optional)	Root/subscription						X					X	X	X		
EventViewerConsumer (Optional with the WMI Tools)	Root/CIMV2						X					X		X	X	X
LogFileEventConsumer (Optional)	Root/subscription						X					X	X	X		
Microsoft WMI Forwarding Consumer Provider	Root/subscription						X						X			
Microsoft WMI Forwarding Consumer Trace Event Provider	Root/subscription					X							X			
Microsoft WMI Forwarding Event Provider	Root/CIMV2					X							X			
NTEventLogEventConsumer	Root/subscription						X					X	X	X		
SMTPEventConsumer (Optional)	Root/subscription						X					X	X	X		
CIMWin32	Root/CIMV2		X	X				X	X	X	X	X	X	X	X	X
CIMWin32A	Root/CIMV2		X	X				X	X	X	X	X	X			
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				
DFSProvider	Root/CIMV2		X	X				X	X	X	X	X				
DskQuotaProvider	Root/CIMV2		X					X	X	X	X	X	X			
HiPerfCooker_v1	Root/WMI		X					X		X		X	X			
HiPerfCooker_v1	Root/CIMV2		X					X		X		X	X			
ieinfo5	Root/CIMV2/Applications/MicrosoftIE		X					X		X		X		X	X	
IIS_PROVIDER	Root/MicrosoftIISv2	X	X	X				X	X	X	X	X				
Microsoft|DSLDAPClassAssociationsProvider|V1.0	Root/directory/LDAP		X					X		X		X	X	X	X	
Microsoft|DSLDAPClassProvider|V1.0	Root/directory/LDAP	X						X		X		X	X	X	X	
Microsoft|DSLDAPInstanceProvider|V1.0	Root/directory/LDAP		X					X	X	X	X	X	X	X	X	
Microsoft|NLB_Provider|V1.0	Root/MicrosoftNLB		X	X				X	X	X	X	X		X		
NlbsNicProv	Root/MicrosoftNLB		X	X				X	X	X		X				
MS_NT_DNS_PROVIDER	Root/MicrosoftDNS		X	X				X	X	X	X	X		X		

Table 1.1 *The Most Important WMI Providers under the Windows Platforms with Their Capabilities (continued)*

Provider Name	Provider Classes	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
MS_NT_EVENTLOG_EVENT_PROVIDER	Root/CIMV2					X						X	X	X	X	X
MS_NT_EVENTLOG_PROVIDER	Root/CIMV2		X	X				X	X	X		X	X	X	X	X
MS_Power_Management_Event_Provider	Root/CIMV2					X						X	X	X	X	X
MS_Shutdown_Event_Provider	Root/CIMV2					X						X	X			
MS_SNMP_CLASS_PROVIDER (Optional)	Root/snmp/localhost	X						X		X		X		X	X	X
MS_SNMP_ENCAPSULATED_EVENT_PROVIDER (Optional)	Root/snmp/localhost					X						X		X	X	X
MS_SNMP_INSTANCE_PROVIDER (Optional)	Root/snmp/localhost		X					X	X	X	X	X		X	X	X
MS_SNMP_REFERENT_EVENT_PROVIDER (Optional)	Root/snmp/localhost					X						X		X	X	X
MS_VIEW_INSTANCE_PROVIDER	Root/MicrosoftIISv2		X	X				X	X	X	X	X		X		
MSIProv	Root/CIMV2		X	X				X	X	X	X	X	X	X	X	X
MSVDS__PROVIDER	Root/CIMV2		X	X				X	X	X	X	X				
MSVSS__PROVIDER	Root/CIMV2		X	X				X	X	X	X	X				
NamedJobObjectActgInfoProv	Root/CIMV2		X					X	X	X	X	X	X			
NamedJobObjectLimitSettingProv	Root/CIMV2		X					X	X	X	X	X	X			
NamedJobObjectProv	Root/CIMV2		X					X	X	X	X	X	X			
NamedJobObjectSecLimitSettingProv	Root/CIMV2		X					X	X	X	X	X	X			
NetDiagProv	Root/CIMV2		X	X				X	X	X	X		X			
NetFrameworkv1Provider	Root/NetFrameworkv1		X	X				X	X	X	X	X				
NT5_GenericPerfProvider_V1	Root/CIMV2		X					X		X		X	X	X	X	
PolicSOM	Root/Policy		X	X				X	X		X	X	X			
PolicStatus	Root/Policy		X					X				X	X			
RegistryEventProvider	Root/DEFAULT					X						X	X	X	X	X
RegPropProv	Root/DEFAULT				X			X	X			X	X	X	X	X
RegProv	Root/DEFAULT		X	X				X	X	X		X	X	X	X	X
RegProv	Root/CIMV2		X					X	X	X		X				

Table 1.1 *The Most Important WMI Providers under the Windows Platforms with Their Capabilities (continued)*

Provider Name	Provider Classes	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
ReplProv1	Root/MicrosoftActiveDirectory		X	X				X		X		X				
RouteEventProvider	Root/CIMV2					X						X	X			
RouteProvider	Root/CIMV2		X					X	X	X	X	X	X			
Rsop Logging Mode Provider	Root/RSOP			X								X	X			
Rsop Planning Mode Provider	Root/RSOP			X								X	X			
SECRCW32	Root/CIMV2		X	X				X	X	X	X	X	X	X	X	X
SessionProvider	Root/CIMV2		X	X				X	X	X	X	X	X			
SystemRestoreProv	Root/Default		X	X				X	X	X			X			
TrustPrv	Root/MicrosoftActiveDirectory		X					X		X		X				
VolumeChangeEvents	Root/CIMV2					X						X	X			
WBEMCORE	Root/CIMV2		X					X	X	X		X	X	X	X	X
Win32_WIN32_COMPUTERSYSTEMWINDOWSPRODUCTACTIVATIONSETTING_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_WINDOWSPRODUCTACTIVATION_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			
Win32_WIN32_PROXY_Prov	Root/CIMV2		X	X				X	X	X	X	X	X			

Table 1.1 *The Most Important WMI Providers under the Windows Platforms with Their Capabilities (continued)*

Provider Name	Provider Classes	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
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			
Win32ClockProvider	Root/CIMV2					X						X	X			
WMIEventProv	Root/WMI					X						X	X	X		
WMIEventProv	Root/MicrosoftNLB					X						X				
WMIPingProvider	Root/CIMV2		X					X		X		X	X			
WMIProv	Root/WMI	X	X	X				X	X	X		X	X	X	X	
WMIProv	Root/MicrosoftNLB	X	X	X				X	X	X		X				
Microsoft WMI Updating Consumer Assoc Provider	Root/subscription		X					X	X	X	X		X			
Microsoft WMI Updating Consumer Event Provider	Root/subscription					X							X			
Microsoft WMI Updating Consumer Provider	Root/subscription						X						X			
Microsoft WMI Template Association Provider	Root/subscription		X					X		X			X			
Microsoft WMI Template Event Provider	Root/subscription					X							X			
Microsoft WMI Template Provider	Root/subscription		X					X	X	X	X		X			
Microsoft WMI Transient Event Provider	Root/subscription					X							X			
Microsoft WMI Transient Provider	Root/subscription		X					X	X	X	X		X			
Microsoft WMI Transient Reboot Event Provider	Root/subscription					X							X			

However, because we use the *provider* qualifier, only the classes representing instances are located. This means that abstract classes, which are often used as parent templates to create subclasses, are not located. In practice, this has no impact on the information we are looking for, because abstract classes do not represent real-world manageable entities (or instances). Since we need to gather information about the classes that represent the real-world entities that we want managed, the abstract classes are useless for our purpose—even if it is good to know their existence in order to have a good knowledge of the implemented object model.

With this script, it is possible to locate all WMI registered providers across all existing namespaces in the CIM repository and find their supported classes. Doing so, the script self-documents the WMI providers available with their characteristics. If, in the future, some applications are adding new providers, a simple run of the script will provide the updated content of the CIM repository with their providers and supported classes.

Next, to gather more information about the classes, a simple run of the **LoadCIMinXL.wsf** script (see Sample 4.32 in the appendix) provides all the class details with its properties and methods.

The result of the script execution (Samples 1.1 through 1.3) under Windows Server 2003 is summarized in Table 1.1. This table shows the most important WMI providers registered in the CIM repository with their respective namespaces and their type (class provider, instance provider, property provider, etc.).

1.4 Helpers for the discovery

Discovering the class capabilities and the instances they provide is a long process. For this, you can use **WMI CIM Studio** or scripts such as those shown in Sample 1.4 ("Listing a single instance of a class with its properties formatted") and Sample 1.5 ("Listing all instances of a class with their properties formatted"). These samples expose some command-line parameters (class name, namespace, userid, and password) to be more generic and usable remotely. The next script, helping us in our discovery and called **GetSingleInstance.Wsf**, exposes the following command-line parameters:

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

Usage: GetSingleInstance.wsf WMIclass WMIInstanceName [/Machine:value] [/
User:value] [/Password:value] [/NameSpace:value]
```

```
Options:

WMIclass : the WMI class to use.
WMIInstanceName : the WMI instance name.
Machine : determine the WMI system to connect to. (default=LocalHost)
User : determine the UserID to perform the remote connection.    (default=none)
Password : determine the password to perform the remote connection. (default=none)
NameSpace : determine the WMI namespace to connect to. (default=Root\CIMv2)
```

The listing of the script is given for information purposes only, since it is pretty simple.

Sample 1.4 *Listing a single instance of a class with its properties formatted*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9: <job>
..:
13:  <runtime>
..:
20:  </runtime>
21:
22:  <script language="VBScript" src=".\Functions\DisplayFormattedPropertyFunction.vbs" />
23:  <script language="VBScript" src=".\Functions\TinyErrorHandler.vbs" />
24:
25:  <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
26:  <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
27:
28:  <script language="VBscript">
29:  <![CDATA[
..:
33:  Const cComputerName = "LocalHost"
34:  Const cWMINameSpace = "root/cimv2"
..:
49:  ' ---------------------------------------------------------------------------
50:  ' Parse the command line parameters
51:  If WScript.Arguments.Unnamed.Count = 0 Or WScript.Arguments.Unnamed.Count = 1 Then
52:     WScript.Arguments.ShowUsage()
53:     WScript.Quit
54:  Else
55:     strWMIclass = WScript.Arguments.Unnamed.Item(0)
56:     strWMIInstance = WScript.Arguments.Unnamed.Item(1)
57:  End If
58:
59:  strUserID = WScript.Arguments.Named("User")
60:  If Len(strUserID) = 0 Then strUserID = ""
61:
62:  strPassword = WScript.Arguments.Named("Password")
63:  If Len(strPassword) = 0 Then strPassword = ""
64:
65:  strComputerName = WScript.Arguments.Named("Machine")
66:  If Len(strComputerName) = 0 Then strComputerName = cComputerName
67:
68:  strWMINameSpace = WScript.Arguments.Named("NameSpace")
69:  If Len(strWMINameSpace) = 0 Then strWMINameSpace = cWMINameSpace
70:  strWMINameSpace = UCase (strWMINameSpace)
71:
```

```
72:  objWMILocator.Security_.AuthenticationLevel = wbemAuthenticationLevelDefault
73:  objWMILocator.Security_.ImpersonationLevel = wbemImpersonationLevelImpersonate
74:
75:  Set objWMIServices = objWMILocator.ConnectServer(strComputerName, strWMINameSpace, _
76:                           strUserID, strPassword)
..:
79:  Set objWMIInstance = objWMIServices.Get (strWMIclass & "=" & strWMIInstance)
..:
82:  Set objWMIPropertySet = objWMIInstance.Properties_
83:  For Each objWMIProperty In objWMIPropertySet
84:   DisplayFormattedProperty objWMIInstance, _
85:                  objWMIProperty.Name, _
86:                  objWMIProperty.Name, _
87:                  Null
88:  Next
..:
94:  ]]>
95:  </script>
96: </job>
97:</package>
```

Sample 1.5 uses the same structure as Sample 1.4. Only the core code logic is displayed. The command-line parameters are:

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

Usage: GetCollectionOfInstances.wsf WMIclass [/Machine:value] [
User:value] [/Password:value] [/NameSpace:value]
Options:

WMIclass  : the WMI class to use.
Machine   : determine the WMI system to connect to. (default=LocalHost)
User      : determine the UserID to perform the remote connection. (default=none)
Password  : determine the password to perform the remote connection. (default=none)
NameSpace : determine the WMI namespace to connect to. (default=Root\CIMv2)
```

Sample 1.5 *Listing all instances of a class with their properties formatted*

```
 1:<?xml version="1.0"?>
 .:
 8:<package>
 9:  <job>
..:
13:    <runtime>
..:
19:    </runtime>
20:
21:    <script language="VBScript" src=".\Functions\DisplayFormattedPropertyFunction.vbs" />
22:    <script language="VBScript" src=".\Functions\TinyErrorHandler.vbs" />
23:
24:    <object progid="WbemScripting.SWbemLocator" id="objWMILocator" reference="true"/>
25:    <object progid="WbemScripting.SWbemDateTime" id="objWMIDateTime" />
26:
27:    <script language="VBscript">
28:    <![CDATA[
..:
```

```
 32:    Const cComputerName = "LocalHost"
 33:    Const cWMINameSpace = "root/cimv2"
 ..:
 72:    Set objWMIServices = objWMILocator.ConnectServer(strComputerName, strWMINameSpace, _
 73:                                                     strUserID, strPassword)
 74:    If Err.Number Then ErrorHandler (Err)
 75:
 76:    Set objWMIInstances = objWMIServices.InstancesOf (strWMIclass)
 77:    If Err.Number Then ErrorHandler (Err)
 78:
 79:    If objWMIInstances.Count Then
 80:       For Each objWMIInstance In objWMIInstances
 81:           WScript.Echo
 82:           Set objWMIPropertySet = objWMIInstance.Properties_
 83:           For Each objWMIProperty In objWMIPropertySet
 84:               DisplayFormattedProperty objWMIInstance, _
 85:                                        objWMIProperty.Name, _
 86:                                        objWMIProperty.Name, _
 87:                                        Null
 88:           Next
 89:           Set objWMIPropertySet = Nothing
 90:       Next
 91:    Else
 92:       WScript.Echo "No instance."
 93:    End If
 ..:
 98:    ]]>
 99:    </script>
100: </job>
101:</package>
```

Instead of using the **DisplayInstanceProperties.vbs** script containing the DisplayInstanceProperties() function (see Sample 4.31 in the appendix), these scripts use the DisplayFormattedPropertyFunction() contained in the **DisplayFormattedPropertyFunction.vbs** script (line 13). This function has the exact same purpose as **DisplayInstanceProperties.vbs** developed previously, but it displays the property content in a different way. As an example:

```
 1:  C:\>GetSingleInstance Win32_Service "'SNMP'"
 2:  Microsoft (R) Windows Script Host Version 5.6
 3:  Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
 4:
 5:  AcceptPause: ........................... FALSE
 6:  AcceptStop: ............................ FALSE
 7:  Caption: ............................... SNMP Service
 8:  CheckPoint: ............................ 0
 9:  CreationclassName: ..................... Win32_Service
10:  Description: ........................... Enables Simple Network Management Protocol (SNMP)
                                             requests to be processed by this computer. If
                                             this service is stopped, the computer will be
                                             unable to process SNMP requests. If this service
                                             is disabled, any services that explicitly depend
                                             on it will fail to start.
11:  DesktopInteract: ....................... FALSE
12:  DisplayName: ........................... SNMP Service
13:  ErrorControl: ......................... Normal
14:  ExitCode: ............................. 1077
15:  *Name: ................................. SNMP
```

```
16:   PathName: ............................. J:\WINDOWS\System32\snmp.exe
17:   ProcessId: ............................. 0
18:   ServiceSpecificExitCode: ............... 0
19:   ServiceType: ........................... Own Process
20:   Started: ............................... FALSE
21:   StartMode: ............................. Manual
22:   StartName: ............................. LocalSystem
23:   State: ................................. Stopped
24:   Status: ................................ OK
25:   SystemCreationclassName: ............... Win32_ComputerSystem
26:   SystemName: ............................ XP-DPEN6400
27:   TagId: ................................. 0
28:   WaitHint: .............................. 0
```

This display formatting is easier to read and can be used to display instance information in a manner similar to some command-line utilities.

The **DisplayFormattedPropertyFunction.vbs** accepts four parameters, as follows:

- The WMI instance object (line 38)
- Any string type that can be used as a label for the property (line 39)
- The property name to extract the value from (line 40)
- A second property name that could be displayed on the same line as the first property (line 41)

When set to null, the second property is not displayed. However, the function does not display the property syntax. In the same way, the property is skipped if its value is null. Moreover, if the property is an array, the function behaves accordingly and transparently. This function will greatly help and ease the display of any class properties for the next samples. The **DisplayFormattedPropertyFunction.vbs** code is given in Sample 1.6 for reference. The WMI scripting technique used was covered in *Understanding WMI Scripting*, Chapter 4.

Sample 1.6 *The DisplayFormattedPropertyFunction.vbs function*

```
 .:
 6:' ------------------------------------------------------------------------------------
 7:Function DisplayFormattedProperty (objWMIInstance, strText, strProperty1, strProperty2)
..:
16:    If Not IsNull (strProperty1) Then
17:       varValue1 = objWMIInstance.Properties_.Item (strProperty1)
18:       If Err.Number Then
19:          varValue1 = strProperty1
20:          Err.Clear
21:       Else
22:          If Ucase (strProperty1) = "TIME_CREATED" Then
23:             objWMIDateTime.SetFileTime (varValue1)
24:             varValue1 = objWMIDateTime.GetVarDate (True) & " (" & objWMIDateTime.Value & ")"
25:          Else
```

```
26:             Select Case objWMIInstance.Properties_.Item (strProperty1).CIMType
27:                    Case wbemCimtypeDatetime
28:                         objWMIDateTime.Value = varValue1
29:                         varValue1 = objWMIDateTime.GetVarDate (False)
30:                    Case wbemCimtypeBoolean
31:                         varValue1 = Ucase (varValue1)
32:                    Case wbemCimtypeObject
33:                         varValue1 = "<OBJECT>"
34:             End Select
35:          End If
36:
37:          boolCIMKey = _
38:              objWMIInstance.Properties_.Item (strProperty1).qualifiers_.Item("key").Value
39:          If Err.Number Then
40:             Err.Clear
41:             boolCIMKey = False
42:          End If
43:          If boolCIMKey Then
44:             If Mid (strText, 1, 1) = Chr (32) Then
45:                intSpace = Len (strText) - Len(LTrim(strText))
46:                strText = Space(intSpace) & "*" & Ltrim(strText)
47:             Else
48:                strText = "*" & strText
49:             End If
50:          End If
51:       End If
52:    Else
53:       varValue1 = strProperty1
54:    End If
55:
56:    If Not IsNull (strProperty2) Then
57:       varValue2 = objWMIInstance.Properties_.Item (strProperty2)
58:       If Err.Number Then
59:          varValue2 = strProperty2
60:          Err.Clear
61:       Else
62:          If Ucase (strProperty2) = "TIME_CREATED" Then
63:             objWMIDateTime.SetFileTime (varValue2)
64:             varValue2 = objWMIDateTime.GetVarDate (True) & " (" & objWMIDateTime.Value & ")"
65:          Else
66:             Select Case objWMIInstance.Properties_.Item (strProperty2).CIMType
67:                    Case wbemCimtypeDatetime
68:                         objWMIDateTime.Value = varValue2
69:                         varValue2 = objWMIDateTime.GetVarDate (False)
70:                    Case wbemCimtypeBoolean
71:                         varValue2 = Ucase (varValue2)
72:                    Case wbemCimtypeObject
73:                         varValue2 = "<OBJECT>"
74:             End Select
75:          End If
76:       End If
77:    Else
78:       varValue2 = strProperty2
79:    End If
80:
81:    If Not IsNull (varValue1) Then
82:       If Not IsNull (varValue2) Then
83:          If IsArray (varValue1) Then
84:             For intIndice=0 To UBound(varValue1)
85:                 If intIndice = 0 Then
```

```
 86:                   WScript.Echo strText & ": " & string (40 - Len(strText), ".") & _
 87:                                " " & varValue1 (intIndice) & _
 88:                                " " & varValue2 (intIndice)
 89:                Else
 90:                   WScript.Echo Space (42) & _
 91:                                " " & varValue1 (intIndice) & _
 92:                                " " & varValue2 (intIndice)
 93:                End if
 94:             Next
 95:          Else
 96:             WScript.Echo strText & ": " & string (40 - Len(strText), ".") & _
 97:                          " " & varValue1 & _
 98:                          " " & varValue2
 99:          End IF
100:       Else
101:          If IsArray (varValue1) Then
102:             For intIndice=0 To UBound(varValue1)
103:                If intIndice = 0 Then
104:                   WScript.Echo strText & ": " & string (40 - Len(strText), ".") & _
105:                                " " & varValue1 (intIndice)
106:                Else
107:                   WScript.Echo Space (42) & _
108:                                " " & varValue1 (intIndice)
109:                End if
110:             Next
111:          Else
112:             WScript.Echo strText & ": " & string (40 - Len(strText), ".") & _
113:                          " " & varValue1
114:          End IF
115:       End IF
116:    End IF
117:
118:    Err.Clear
119:
120:End Function
```

For now, we have four scripts that can be used as utilities to help our understanding and WMI discovery:

- **LoadCIMinXL.wsf**: a Windows script file self-documenting the classes available from the CIM repository in an Excel sheet (see Sample 4.32 in the appendix).
- **LocateProviders.wsf**: to locate the WMI provider registration instances with the supported classes (Samples 1.1 through 1.3).
- **GetSingleInstance.wsf**: to list a single instance of a class with its properties formatted (Sample 1.4).
- **GetCollectionOfInstances.wsf**: to list all instances of a class with their properties formatted (Sample 1.5).

With the help of these scripts, our next purpose is to develop small utilities based on the WMI scripting techniques and the class capabilities.

1.5 Summary

Because a class is a template that represents a manageable entity from the real world, made accessible with a provider, mastering the provider capabilities with its classes is a determinant factor when developing applications on top of WMI.

Although our class classification is made by WMI providers, Microsoft, in its Platform SDK, classifies the classes by role. This classification regroups the classes that are related to the same parts of the system independently of the WMI provider supporting them. This difference comes from the fact that Microsoft estimates that it is not necessary to know and understand the provider implementation to work with the classes they support. This statement is true in a sense, but having a rough idea of the provider capabilities, such as determining if it is implemented as an event provider or an instance provider, can help people to determine how a WQL event query must be formulated (use of the **WITHIN** statement for dynamic instance classes or not using the **WITHIN** statement when an extrinsic event class is referenced in the query, as it is supported by an event provider).

When installing Windows Server 2003, WMI is part of the installation and it comes with an important number of WMI providers and classes. Although we can list the classes available from the providers, it is impossible to list each class properties in a single book, but the helper scripts (**LoadCIMinXL.wsf**, **LocateProviders.wsf**, **GetSingleInstance.wsf**, **GetCollectionOfInstances.wsf**) can be used as tools to complete the discovery. From a manageability perspective, Windows Server 2003 represents a major step for the Enterprise because it offers much more WMI management capabilities than Windows 2000. Besides the WMI functionalities part of the Operating System installation, Windows applications such as Exchange 2000, SQL Server 2000, and HP OpenView Operations for Windows (OVOW), to name a few, bring their own providers and enhance the WMI capabilities. This means that there is no limit to the WMI discovery, since each new piece of software can extend the WMI capabilities. This is where the helper scripts become interesting, since they can help you discover information not necessarily documented in an SDK.

In this chapter, we learned that the WMI provider discovery technique helps us to understand the WMI provider types available, to structure their roles and the classes they support, since each provider brings its own

set of capabilities with a certain number of classes. Now, the next step is to examine each of them with their own set of classes with their capabilities. Therefore, in the next chapter, we will pursue the discovery by exploring the *Win32* providers, which serve as a great example since they support the largest number of classes available from WMI.