2

Starting with WMI

2.1 Objective

After reviewing WSH, we can start the exploration of Windows Management Instrumentation (WMI). Of course, before scripting on top of WMI, we must learn what WMI is. This implies a theoretical approach to the technology. Like any technology, WMI is a kind of "computing environment" in which you must be able to navigate. It is important to understand how WMI is implemented under Windows and its key components. WMI refers to some standards defined by a committee called the Distributed Management Task Force (DMTF). We will see what this committee is and the initiatives behind it. Of course, the purpose of this chapter is not to delve into the discovery of the DMTF standards, but to provide the necessary background information to understand WMI. Throughout this chapter, we discover the WMI architecture and review its concepts and terms. We explore the WMI database and examine the definition of its elements and how they are related. As we progress through the chapter, we build an alerting system that helps to illustrate the various concepts as they are explained.

2.2 What is the DMTF?

To understand Windows Management Instrumentation, we must first look at the DMTF and its relation to Web-based Enterprise Management (WBEM) and WMI. The DMTF was founded in 1992 as the Desktop Management Task Force, and it focused on standards for managing desktop PCs. With the evolution of the industry, the management of several single entities at the enterprise level was challenging. To reflect this evolution, the DMTF kept its acronym, but the meaning was changed to the Distributed Management Task Force. Today, the DMTF's mission is to lead the devel-

opment of management standards for distributed desktop, network, enterprise, and Internet environments. The DMTF is composed of several computer industry key players such as 3Com, Hewlett-Packard, Cisco, Microsoft, SUN, Intel, IBM/Tivoli Systems, Novell, and others. Beside these computer companies, there are also smaller software companies and universities (called academic members). All participants in the DMTF have made different contributions to the initiative based on their DMTF membership. The list of members participating in the DMTF initiative is quite impressive. For a complete list go to http://www.dmtf.org. It is important to note that the DMTF is a not-for-profit organization!

The DMTF started many unifying management initiatives while focusing on distributed management. Here is a brief overview of these initiatives:

- *Desktop Management Interface (DMI):* DMI is a standard framework for managing and tracking components in desktop PCs, notebooks, or servers. DMI is currently the only desktop standard in adoption for today's PCs. Version 2.0 of the DMI specification was released by the DMTF in April 1996. You can go to http://www.dmtf.org/standards/index.php for more information.
- *Web-based Enterprise Management (WBEM):* WBEM is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. The DMTF has developed a core set of standards that make up WBEM, which includes a data model, an encoding specification, and a transport mechanism. You can go to http://www.dmtf.org/standards/index.php for more information, but we revisit WBEM throughout the chapter as WMI implements the WBEM standards.
- *Common Information Model (CIM):* CIM is a common data model of an implementation-neutral schema to describe overall management information in a network/enterprise environment. CIM includes a specification and a schema. The specification defines the details for integration with other management models such as SNMP's MIBs or CMIP. The schema provides the data model descriptions. You can go to http://www.dmtf.org/standards/index.php for more information, but we revisit it throughout this chapter as WMI uses CIM extensively.
- *Directory Enabled Networks (DEN):* The DEN specification is designed to provide the building blocks for more intelligent networks by mapping users to network services and mapping business criteria to the delivery of network services. This enables applications and

services to leverage the network infrastructure transparently on behalf of the user, empower end-to-end services, and support distributed network-wide service creation, provisioning, and management. DEN specifies a common data model with LDAP mappings from CIM to X.500. This provides a template for exchanging information and enables vendors to share a common definition of a device, application, or service, and allows for extensions. DEN also provides a level of interoperability with WBEM-based solutions. Both DEN and WBEM are built on the CIM specifications. You can go to http://www.dmtf.org/standards/index.php for more information.

- *Service Incident Standard (SIS) and Solution Exchange Standard (SES):* The DMTF has worked closely with the Customer Support Consortium in the joint development of standards for customer support and help-desk applications by providing common data models for incident and solution exchange. For more information, you can go to http://www.dmtf.org/standards/index.php and http://www.serviceinnovation.org.
- *Alert Standard Forum (ASF):* ASF is a standard for managing pre-OS environments. The term *system manageability* represents a wide range of technologies. This involves access to and control of remote systems. The term *remote systems* includes systems with OS-present (e.g., any computer system such as Unix systems, Windows systems, mainframes) and with OS-absent (e.g., printers, hubs). The DMTF defines DMI and CIM interfaces to serve clients' OS-present environments. The ASF specification defines remote control and alerting interfaces that best serve clients' OS-absent environments.

Why did the DMTF take these initiatives? We know a computer network is composed of many devices, computers, peripherals, and the like. These components are themselves manageable in various ways. Although computers from various manufacturers may differ, they all have some common characteristics. Every computer has network devices, hard disks, controllers, tapes, memory, and processors, regardless of the manufacturer. In the same way, operating systems also have common characteristics, such as the memory available to run applications, the processes running, disk space available, users configured in the system, print queues and print jobs, and network adapters status. All these things represent a collection of objects or entities that require particular care when some management tasks are executed. As one would expect, you do not manage a hard disk in the same way as you manage a user or a print job!

In the early stages of the computer industry, each time it was required to get access to a component (to retrieve some information such as its status, for instance), it was necessary to know how to play with that component because the manufacturer implemented a proprietary access method. So, the fact that the component was part of an IBM system or a HP system implied that the method used to retrieve the desired information was not the same. Moreover, the information representing the relevant data was likely to be coded in a format specific to the manufacturer. The access methods were based on a collection of functions often provided by the computer manufacturer (generally part of the BIOS). This manufacturer-specific implementation required an adapted knowledge of the considered computer to retrieve the information. Besides the hardware differences, the operating systems used by computers determined the method used to retrieve the desired information. All these differences increased the complexity and resulted in poor interoperability from a management perspective. The real world rarely uses one computer type with one operating system. Over the years, many initiatives were developed to collect management information—for example, Simple Network Management Protocol (SNMP), Desktop Management Information (DMI), and Common Management Information Protocol (CMIP). As a result, a solution was needed to deal with this mix of technologies and differences with one main objective: making the real world more manageable from a global perspective. This is exactly the purpose of the DMTF.

2.3 What is WBEM?

Since 1996, the goal has been to provide a standard that defines how these entities can be managed and provide system managers with a low cost solution for their management needs without having to learn all the details and specificities from various computer manufacturers and developers. One of the initiatives of the DMTF is Web-Based Enterprise Management (WBEM). WBEM is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. Unifying means not only having a common access to the manageable entities (HTTP), but also having a common way to represent these manageable entities with its characteristics (CIM and xmlCIM).

WBEM can be summarized as a set of standards that define abstraction layers that hide the complexity of accessing management information.

2.4 What is CIM?

The DMTF delivers to the industry a core set of standards that creates WBEM. As previously mentioned, one of the key elements is to have a common representation of the manageable entities of the real world. This implies the creation of a data model that everyone can use. All the components that must be managed in a computer network (i.e., computers, disks, printers, memory, CPU, processes) can be seen as objects. All these objects represent a set of information that could be described in a data model. To produce this data model, the WBEM initiative includes the CIM standard. CIM is a conceptual information model describing management information that is not bound to a particular implementation. For example, a disk from one computer manufacturer has a set of characteristics common to any disk provided by any other manufacturer. Even if they are not the same, all disks have a capacity, an access speed, and a certain number of sectors and tracks (or cylinders). Even if a computer component has some particularities, there is always a common set of characteristics that you will retrieve in every implementation. A disk is a disk, just as an adapter is an adapter regardless of whether it is from IBM or HP. All adapters use I/O addresses, and a bus connector type. It is possible to delve further and differentiate between adapters. We may have modem adapters, network adapters, video adapters, among others. Again, a video adapter from one manufacturer is different from a video adapter from another manufacturer, but they share a common set of characteristics, such as the refresh rate, resolution, and number of colors supported. By examining carefully each manageable component available in the computer industry, it is possible to create a model, such as the CIM data model, that describes their characteristics in a generic way. In a fully CIM-compliant world, it is possible to build applications using management data from a variety of sources and different management systems. The management data is collected, stored, and analyzed using a common format. The CIM standard has two parts: the CIM specification and the CIM Schema.

2.4.1 The CIM specification

The CIM specification defines a data model for the CIM standard. This describes the way the data model is encoded (i.e., how an object representing a disk is coded) by using Managed Object Format (MOF) files or Extended Markup Language (xmlCIM). A MOF file is an ASCII file that uses a specific syntax and language definition to code CIM definitions. We

will revisit MOF files in Section 2.5. The xmlCIM encoding defines XML elements (XML tags) written in a Document Type Definition (DTD), which can be used to represent objects (i.e., disks, memory, adapters, and software components, such as files, volumes, processes) in an XML format. The transport mechanism over HTTP of the XML representation allows implementations of CIM to interoperate in an open and standardized manner. This means that System A of one constructor will be able to interoperate with System B of another constructor.

Note that under Microsoft Windows.NET Server and SUN OS 8.0 (both implement a version of WBEM), it is possible to represent CIM elements in XML. However, at the time of this writing, it is not possible to exchange CIM XML information over HTTP. In Chapter 5 we will see how to get an XML representation of a CIM object.

The CIM specification also provides naming and mapping techniques to other management models, such as Simple Network Management Protocol Management Information Base (SNMP MIB), Desktop Management Interface (DMI), and Common Management Information Protocol (CMIP).

The specification also defines a *metaschema*. The metaschema is the formal definition of the data model used. As CIM represents miscellaneous manageable components, such as disks, software, and processes, the data model needs some structure to represent these manageable objects. To implement a data model, some elements or building blocks must be used. These elements can help to define the nature of things. For example, a disk is a disk, and it is not an adapter or a process. In the CIM data model this is represented as a *class*. Each object has a specific nature and specific properties. For example, a disk has a capacity, speed, and geometry. On the other hand, a network adapter has a media type and a link speed. These characteristics are called *properties*, which represent another element of the CIM data model. A video card has a resolution property that can be modified by specific actions. In the CIM data model, these actions are known as *methods* and are exposed by the object itself. The metaschema CIM specification defines elements like classes, properties, and methods. Classes in CIM can be of different types (e.g., indications and associations). The same approach can be applied to the properties, as they can be of a type called *references*. The CIM specification is the definition of all the terms, methods and concepts used to produce the data model used by CIM. Like any specification, these documents are not always easy to read, but if you would like more information about the CIM specification you can go to http://www.dmtf.org.

2.4.2 The CIM Schema

The CIM Schema is the key component that represents the data model. It contains the management schemas that represent the building blocks to manage computer systems and applications. The CIM Schema represents a collection of objects with different relationships. The CIM Object Model represents objects that are available in the real world. It has nothing to do with object-oriented programming, as object-oriented programming does not necessarily represent things from the real world. It is important to understand that an object that represents a manageable entity exists in the CIM object model. The CIM schema uses specific terminology to define its elements. A summary of these elements includes the following:

- **Classes** represent the nature of an object. For example, a computer system can use several disks with different characteristics, but all the disks will have a common representation defining their nature; in the case of this example, the class will be *disk*. However, for the computer, as the nature of a computer is different from the nature of a disk, we will have another class called *computer*. Therefore, a class is a template defining the nature of an object. Moreover, it is also a template defining a list of properties typical to the nature of the object it represents. For example, in the case of a *disk* class, the class will have properties such as *disk size* or *access time*.
- **Subclasses** are a subtype of a class (the latter is called a *superclass* or *parent class*). For example, a manageable component can be a medium of any type. Therefore, the CIM representation has a *media* class that represents media in general. As media exist in various forms such as disks, CD-ROMs, tape drives, and optical drives, the *media* superclass will have subclasses representing the disks, CD-ROM, tape drives, and optical drives. In this example, we will have the *disk*, *cdrom* and *tape* classes defined as subclasses of the *media* superclass. Because there is a notion of parent-child between the *media* superclass—and, for example, the *disk* class—there is a notion of inheritance between the classes. Actually, a subclass inherits the characteristics of its superclass. When a class is built from a superclass, we could also say the subclass is derived from the superclass. Therefore, a subclass is a template defining the nature of an object, but it defines the nature of the object more specifically.
- **Instances** represent objects or items of the real world. An instance is always created from the class representing the template for the real-

world objects. For example, a disk instance represents a real disk where its template is defined by the *disk* class.

- **Properties** are the attributes of real-world objects. For example, an attribute can be the size of a disk, the process ID of a process, the name of installed software. The definition of the properties is made in the class definition. When an instance of a real-world object is created, the values representing the characteristics of that object are stored in the instance properties as defined by the corresponding class. Therefore, in the case of a disk, we have a *disk* class defining the template representing the disks with a set of properties common to all disks. Once an instance of a disk is created, the characteristics of a real-world disk are stored in the properties of the instance created from the *disk* class.
- **Relationships** are elements used in the CIM schema to associate different objects. For example, a disk volume cannot exist if it is not associated with a disk partition and a disk partition cannot exist without a disk. We will see further that there are different types of relationships.

Note that the concept of data modeling can be dedicated to other universes than the computing universe, but CIM represents manageable software and hardware items only. To present objects of a data model, a specific language is used: the Unified Modeling Language (UML). UML uses boxes and lines to represent objects and relationships. Let's take an easy example not related to the computing world, something that everybody knows, something that everybody has tasted at some stage—a cheeseburger! A cheeseburger is composed of various things: bread, beef, a gherkin, cheese, and ketchup. How do we represent a cheeseburger with UML? We must start at the highest generic level, which means that we will have a representation of the various types of ingredients as shown in Figure 2.1. These ingredients will be the superclasses.

Figure 2.1 *The generic ingredients of the cheeseburger.*

We have generic classes representing the family of each ingredient; for example, *meat* represents the beef, and *vegetable* represents the gherkin. In UML, the classes are represented in a rectangle. Each class may have properties. Properties are represented in a second region of the rectangle (see Figure 2.2).

Figure 2.2 *The generic ingredients of the cheeseburger with its properties.*

To check the validity of the ingredients, the data model can provide a method to determine whether an aliment is still good to eat. A third section in the rectangle represents methods (see Figure 2.3).

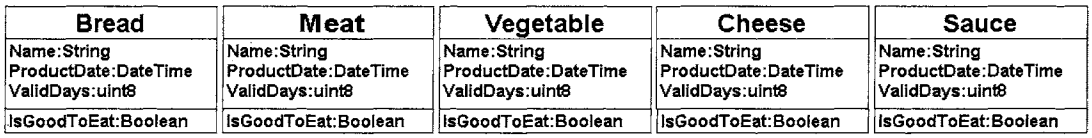

Figure 2.3 *The generic ingredients of the cheeseburger with its properties and method.*

There are many different types of meat such as *beef*, *pork*, or *lamb*. In this case, we have subclasses. The *meat* class is a superclass of the *beef*, the *pork*, and the *lamb* classes. In other words, the *beef*, the *pork*, and the *lamb* classes are subclasses of the *meat* class. We can also say that the *beef*, the *pork* and the *lamb* classes are derived from the *meat* class, which makes those classes inheritors of the properties of the *meat* class. UML uses a line to represent the relationship between the superclasses and the subclasses. An arrowhead points to the superclass to indicate that it is the parent class (see Figure 2.4).

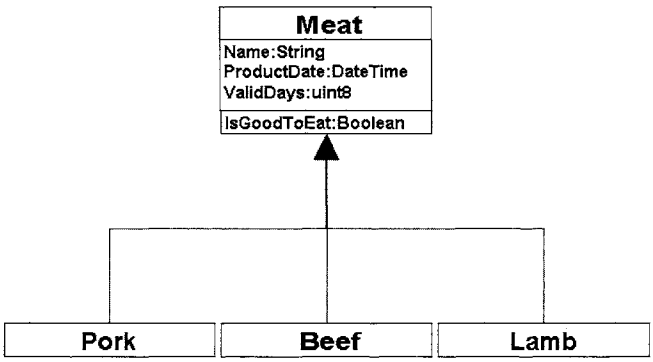

Figure 2.4 *The meat generic ingredient derivation.*

Of course, we have the same relationship for the vegetable (where we may have subclasses such as *salad*, *gherkin*, *onions*, *potatoes*, etc.), the sauces (where we may have subclasses such as *ketchup*, *mayonnaise*, *mustard*, etc.) For the *bread* and the *cheese* superclasses, we have a *toasted bun* and a *processed cheese slice* as subclasses, respectively. The UML representation for each of the subclasses is shown in Figure 2.5.

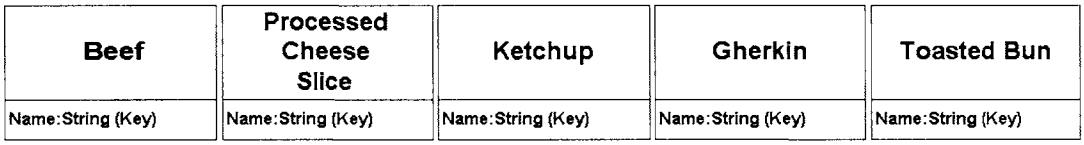

Figure 2.5 *The base ingredients of the cheeseburger.*

Note that one property is already defined in the subclasses. As mentioned, subclasses can inherit the properties from their superclasses and, at the same time, new properties can be added to the subclass. Figure 2.5 shows that an added property can have the same name as a property defined in the superclass. Therefore, in the CIM representation, a subclass can override a property defined in its superclass. These possibilities are defined by schema elements known as qualifiers.

To produce a cheeseburger, we must combine all these ingredients together; for this, we create some associations. Figure 2.6 shows the associations required for a cheeseburger. The associations illustrate that a cheeseburger is composed of beef, processed cheese slice, ketchup, gherkin, and toasted bun.

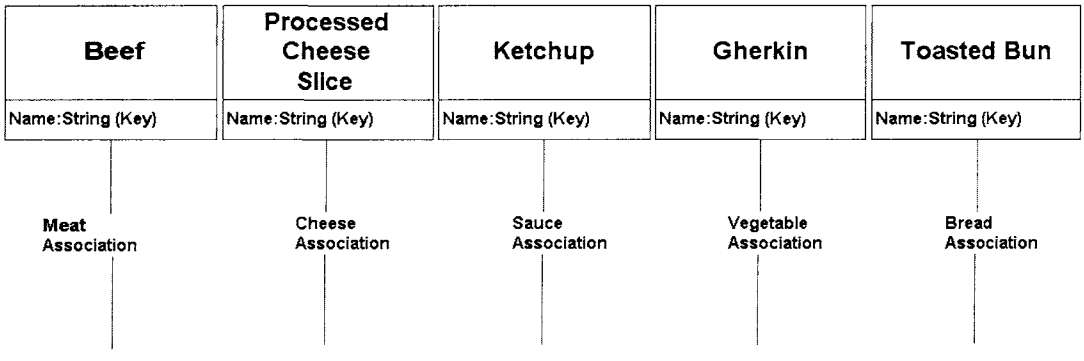

Figure 2.6 *The association of the base ingredients of the cheeseburger.*

Associations are represented by lines with the name of the association usually placed near the center of the line. In CIM, associations are classes and always have a role. Roles are defined in a set of properties known as references, which are valid only in the context of an association. For example, to produce a cheeseburger, we must have at least one toasted bun, one gherkin, an amount of ketchup, one processed cheese slice, and one or two slices of beef (although two slices of beef will produce a double cheeseburger, but let's say for simplicity that it is still a cheeseburger). Each item plays the role of ingredient within its own context; each item plays a particular ingredient role. With regard to the ingredients, we have five roles: *MeatAssociation*, *CheeseAssociation*, *SauceAssociation*, *VegetableAssociation*, and *BreadAssociation*.

Each role is part of an association. As soon as the ingredients classes are associated via the defined references, we have a cheeseburger. Figure 2.7 shows that each association has references. For example, the association *MeatAssociation* has the references *Burger* and *MeatIngredient*. The *cheeseburger* is a class, which has some associations with the various ingredients. Moreover, the *cheeseburger* may also have some properties such as its weight.

CIM also has the notion of an aggregation relationship in which one entity is made up of the aggregation of some number of other entities. An aggregation is just a variation on the concept of an association. For example,

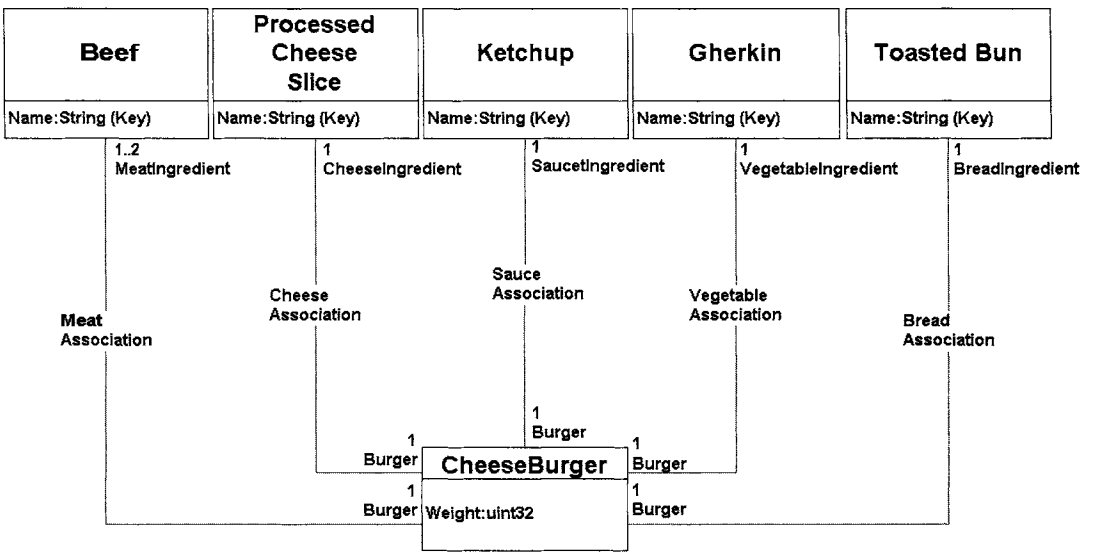

Figure 2.7 *The cheeseburger constitution with the association of the base ingredients.*

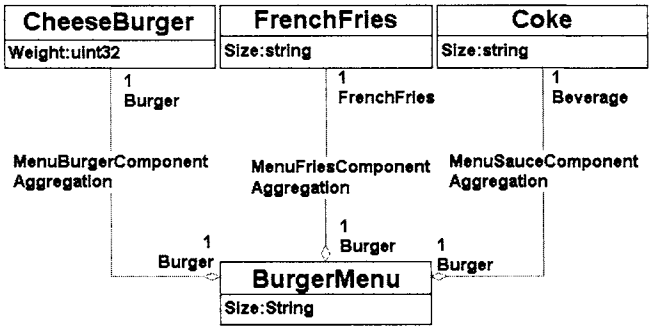

Figure 2.8 *The burger menu constitution with the product aggregations.*

if we aggregate the cheeseburger with some french fries and a Coke, we get a burger menu (see Figure 2.8).

Aggregations are represented, like associations, by lines with the name of the aggregation placed near the center of the line. To distinguish an association from an aggregation, the aggregation terminates on one end with a lozenge.

This example shows the concept behind the CIM representation of real objects and how the abstraction is organized. The CIM data model starts always from the most general representation and tends to define things in a more precise way by using associations and inheritances between objects.

Microsoft delivers some tools to work with CIM. One of the tools used to explore the CIM repository is called **WMI CIM Studio**. It available as a separated download for Windows 2000, Windows XP, and Windows.NET Server from http://www.microsoft.com/downloads/release.asp?ReleaseID=40804. We will revisit this tool in more detail in Section 2.8.5.2. Figure 2.9 shows a sample screen shot of the **WMI CIM Studio**.

The tree view is expanded at the level of the *CIM_ManagedSystemElement* class (left pane). Below the *CIM_ManagedSystemElement* class we have a collection of classes, the *CIM_LogicalElement* class being the first. The *CIM_LogicalElement* subclass contains another set of classes, such as, *CIM_System*, *CIM_Service*, and *CIM_LogicalDevice* classes. When we arrive at the *CIM_LogicalDevice* class level, we can see that the class definition is becoming more and more precise because we started from a class called *CIM_ManagedSystemElement* to arrive at *CIM_LogicalDevice*. In the same way, below the *CIM_LogicalDevice*, we have a collection of classes representing the logical devices, such as the *CIM_MediaAccessDevice*, *CIM_CoolingDevice*, *CIM_Controller*, and *CIM_NetworkAdapter* classes. To complete the picture, if we look below the *CIM_MediaAccessDevice* class, we see classes such as *CIM_CDROMDrive*, *CIM_DiskDrive*, *CIM_TapeDrive*, and

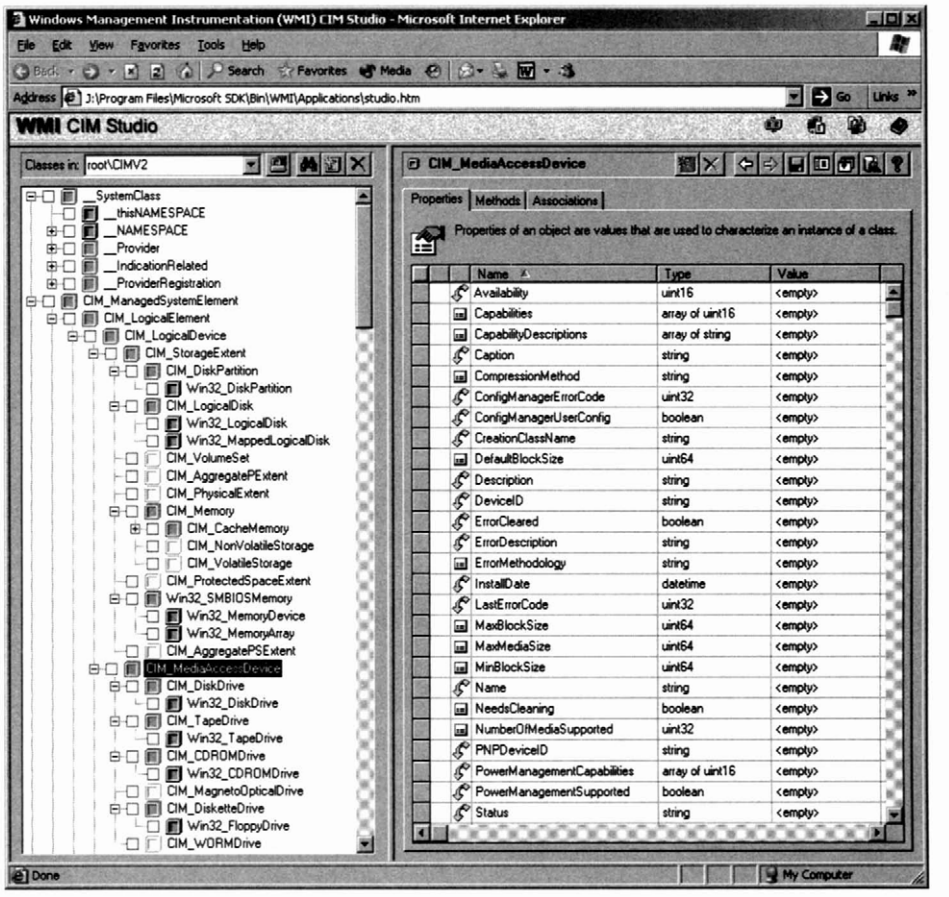

Figure 2.9 *WMI CIM Studio.*

CIM_DisketteDrive classes. The further we navigate down the tree, the definition of the CIM class tends to be more and more precise in defining its purpose

2.4.3 The CIM structure

A Core Model and a Common Model constitute the CIM schema. Since CIM starts from the most generic definition of the manageable objects in the computing world, it defines the Core Schema and the appropriate Common Schema area. After the CIM Core Schema come the Extension Schemas. The number of Extension Schemas depends on the CIM Schema version. Each new release of the schema added a new Extension Schema (see Figure 2.10).

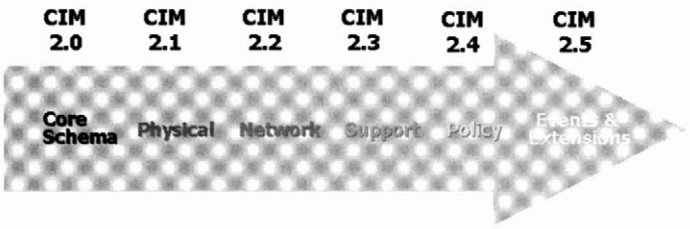

Figure 2.10
The CIM versions.

DMTF recently released version 2.7 of the CIM Schema. Regardless of the version, the structure always determines three levels of classes:

1. **Classes representing managed objects that apply to all areas of management.** These classes provide a basic vocabulary for analyzing and describing managed systems. They are part of the Core Model. The Core Model is not specific to any platform. In Figure 2.9, the *CIM_ManagedSystemElement* class is a class of the Core Schema. This class does not relate to any particular real managed object or domain.

2. **Classes representing managed objects that apply to specific management areas, but are independent of a particular implementation or technology.** These classes are part of what is called the Common Model, which is an extension of the Core Model. The management areas included in the Common Model are System, Application, Physical, Device, Network, Metric, Supports, and User. In Figure 2.9, *CIM_LogicalDevice* class and the subclass *CIM_MediaAccessDevice* are two sample classes of the Common Model.

3. **Classes representing managed objects that are technology-specific additions to the common model.** These classes typically apply to specific platforms such as UNIX or Windows. For example, in Figure 2.9, *Win32_CDROMDrive* and *Win32_FloppyDrive* are two Microsoft classes from the extended CIM Schema that represent a CD-ROM and a floppy drive, respectively.

Figure 2.11 shows the logical representation of the CIM structure, where we have the Core Schema in the center with the Common Models acting as an extension of the Core Schema. Besides these two CIM general representations, we have the technology-specific additions such as WMI.

Applied to the computing environment, and to be CIM compliant, a product has to implement its representation in the same way. This implies that the Microsoft implementation of the CIM Schema will follow this

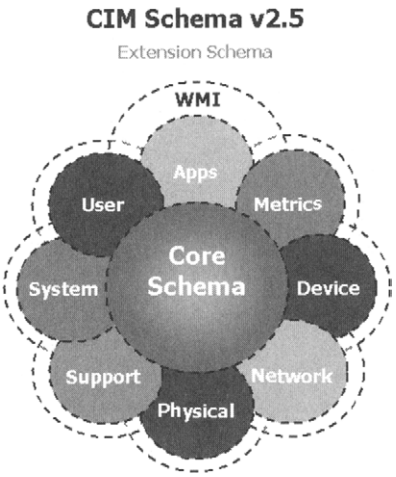

Figure 2.11 *The Core Schema and the Common Schema.*

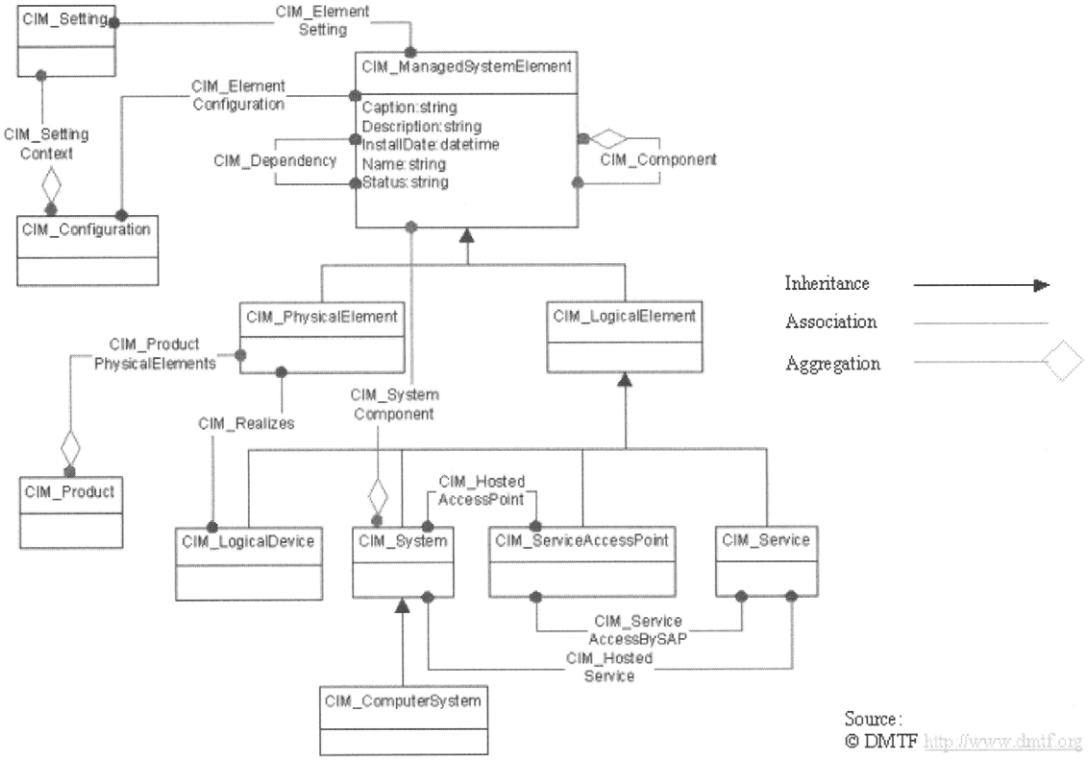

Figure 2.12 *The Core Schema UML representation.*

implementation as designed by the DMTF. A schema has a name and contains a collection of classes. The Core Schema contains a top-level set of classes applicable to all management domains. Figure 2.12 shows a partial UML representation of the CIM Core Schema. It is possible to download the complete UML representation of the CIM Core Schema and the CIM Common Schemas from http://www.dmtf.org. All schemas are greatly detailed and available in Visio or Adobe Acrobat file formats.

By observing Figure 2.12, you will notice that the UML representation shows clearly that the *CIM_LogicalElement* class has an inheritance with the *CIM_ManagedSystemElement* class. Next, the *CIM_LogicalElement* class is derived to produce the *CIM_System*, *CIM_LogicalDevice*, *CIM_Service* and *CIM_ServiceAccessPoint* classes. The UML representation matches the **WMI CIM Studio** view shown in Figure 2.9.

2.4.4 The CIM Schema elements

Now let's review the CIM Schema elements as they are named and used in CIM. While examining the different elements you can refer to Figure 2.13. This figure shows a logical representation of the relationship between the different terms we will examine in subsequent paragraphs. Keep in mind that the figure is just an abstract representation of the elements and does not show how the things are actually implemented or linked.

It is very important to have a clear idea of these terms and the things they represent, as they will be used in this book in many different situations. Failure to understand these terms will be a serious handicap when we dive deeper into WMI.

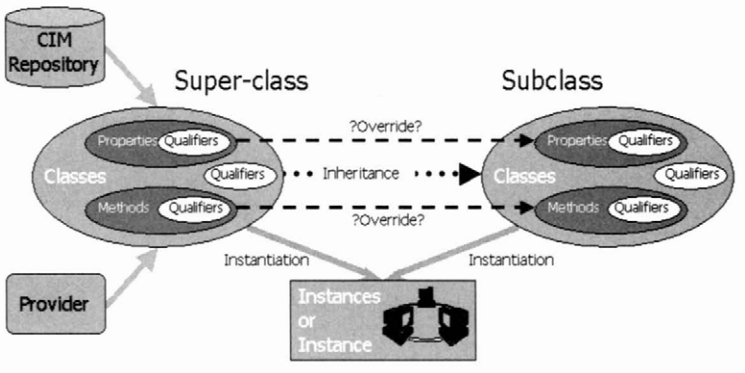

Figure 2.13 *A theoretical representation of the CIM Schema element relationships.*

2.4.4.1 Namespace

A namespace is a portion of a schema. It is also a logical unit to group classes and instances inside the CIM repository. The notion of a namespace also defines the notion of scope and visibility for classes and instances. CIM supports namespace nesting to produce a kind of hierarchy. Even if namespaces can be arranged in a form of hierarchy, they don't play any role in class inheritances. Generally, managed objects of a particular environment are grouped beneath one namespace. Figure 2.14 shows the namespaces available in the

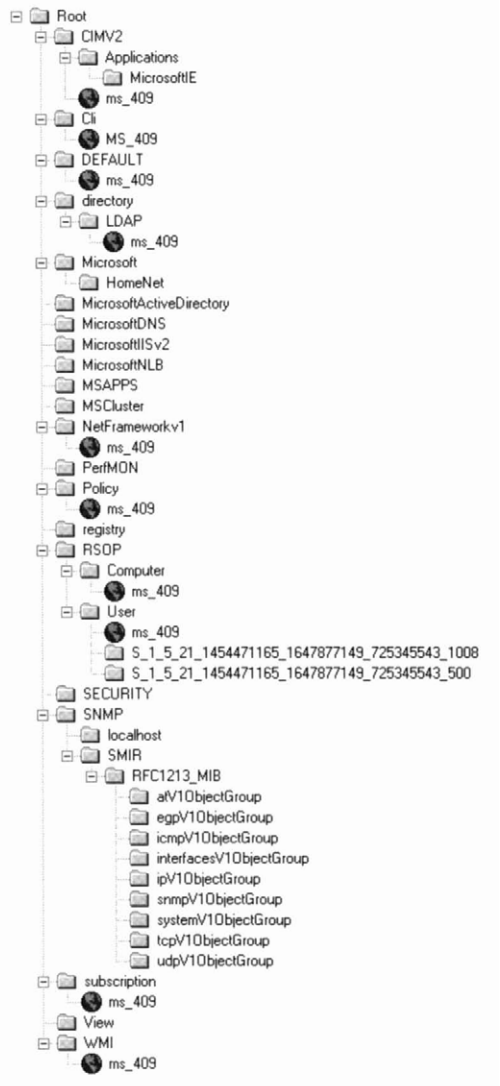

Figure 2.14 *The CIM namespaces under Windows.NET Server as visible from the WMI MMC snap-in.*

CIM repository under Windows.NET Server. For example, the **Root\CIMv2** namespace contains most classes defined by Microsoft to represent the Windows world manageable entities (see Figure 2.14).

By default, all namespaces in the CIM repository contain language-neutral classes. However, some namespaces, such as the **Root\CIMv2** namespace, have a child namespace called **ms_409**, as shown in Figure 2.14. This namespace contains the U.S. version (code page 409) of the language-neutral classes contained in the **Root\CIMv2** namespace. In conclusion, besides the language-neutral classes, WMI allows storage of multiple localized versions of classes, where the ms_<CodePage> namespace contains the language-specific version of the classes (i.e., English, German, French). The language-specific class definitions are always stored in a child namespace beneath the namespace that contains the language-neutral class definition.

With **WMI CIM Studio**, it is possible to view the namespace instances. They are represented as instances of the *__Namespace* system class. When an instance of the *__Namespace* class is created, a new namespace is created. When an instance of the *__Namespace* class is deleted, a namespace is deleted. A list of existing namespaces under Windows.NET Server is available in Table 2.1.

Table 2.1 *The Windows.NET Server WMI Namespaces*

Root
Root/CIMV2
Root/CIMV2/Applications
Root/CIMV2/Applications/MicrosoftIE
Root/CIMV2/ms_409
Root/Cli
Root/Cli/MS_409
Root/DEFAULT
Root/DEFAULT/ms_409
Root/directory
Root/directory/LDAP
Root/directory/LDAP/ms_409
Root/Microsoft
Root/Microsoft/HomeNet
Root/MicrosoftActiveDirectory

Table 2.1 *The Windows.NET Server WMI Namespaces (continued)*

Root/MicrosoftDNS
Root/MicrosoftIISv2
Root/MicrosoftNLB
Root/MSAPPS
Root/MSCluster
Root/NetFrameworkv1
Root/NetFrameworkv1/ms_409
Root/PerfMON
Root/Policy
Root/Policy/ms_409
Root/registry
Root/RSOP
Root/RSOP/Computer
Root/RSOP/Computer/ms_409
Root/RSOP/User
Root/RSOP/User/ms_409
Root/SECURITY
Root/SNMP
Root/SNMP/localhost
Root/subscription
Root/subscription/ms_409
Root/View
Root/WMI
Root/WMI/ms_409

2.4.4.2 Classes

To recap, a class defines the basic unit of management and represents a template for a managed object. A class may have properties (characteristics) and methods (to perform some actions). Classes represent the nature of things, but just like things that exist, classes also have instances. An instance is the representation of a real-world managed object that belongs to a particular class and contains real data. For example, if a class defines a disk, we have one class type for the disk. In the real world, we may have many disks in one computer, which means that the *disk* class may have many instances.

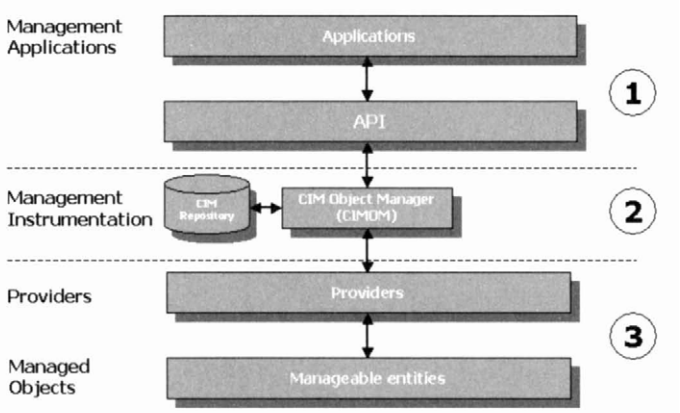

Figure 2.15 *The CIM Object Manager positioning among other WMI components.*

Under WBEM, classes can be static or dynamic. Static classes are stored in the CIM repository and retrieved on request by the CIM Object Manager (CIMOM). The CIMOM is a collection of application programming interfaces (API) that allow applications to interact with the CIM repository. The dynamic classes are generated by a WMI provider. A provider is a software component that communicates with the managed object to access its data. (See Figure 2.15.)

Besides the implementation type of the class (in the CIM repository for static classes and via a provider for dynamic classes), classes can also be qualified in different ways:

- **An abstract class is a class that does not have an instance of its own.** This class type can be derived to produce another class that may have an instance. An abstract class is a kind of template to produce other classes by inheritance. For example, the *CIM_ManagedSystemElement* class shown in Figure 2.9 is an abstract class.
- **System classes are a collection of predefined classes based on CIM and included in every namespace (a namespace is a portion of the schema).** In the Microsoft implementation, system classes are used to support the WMI activities such as provider registration, security, and event notification. System classes follow a naming convention that consists of a double-underscore followed by the class name. For example, the *__Systemclass* in Figure 2.9 is an abstract system class having some system subclasses such as *__provider* or *__NAMESPACE*.
- **A singleton class is a class that supports one instance of the managed object it represents.** For example, the Microsoft WMI class *Win32_WMISetting* is a singleton class as there is only one instance of the WMI Settings in a Windows system.

- **Classes can participate in associations.** In such a case, the produced associations (which can be another class) will have one or more references pointing to the classes participating in that association. For example, the *CIM_LogicalDiskBasedOnPartition* class in Figure 2.16 is an association class linking the *CIM_DiskPartition* and the *CIM_LogicalDisk* classes. We clearly realize the sense of such association because a disk partition cannot exist without a disk.
- **Classes can also be localized.** A class definition is (generally) a language-neutral class, but it may also exist in multiple language-specific classes (i.e., English, German, French). For example, the *Win32_Service* class, which is the class representing a Windows NT service, exists in a language-neutral form (in the **Root\CIMv2** namespace) and in an English U.S. (code page 409) form (in the

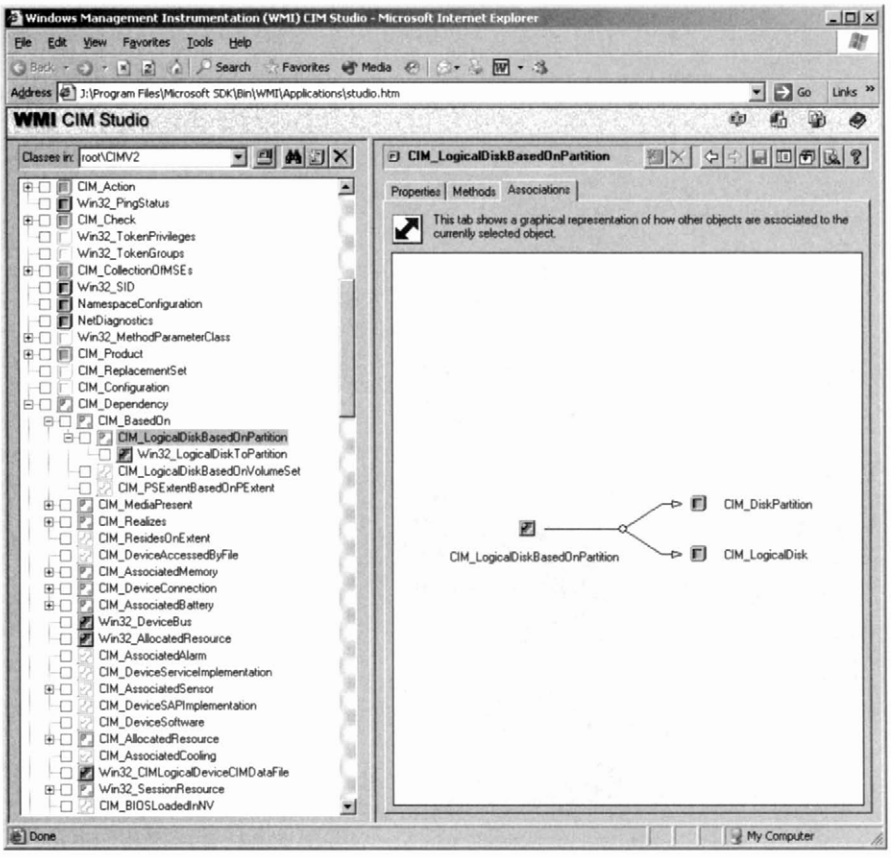

Figure 2.16 *The WMI CIM Studio tool.*

Root\CIMv2\ms_409 namespace). In such cases, a localized class will also be referred as an amended class. A localized class contains an amended qualifier in its definition. Briefly, a qualifier is an attribute of a class, property, or method definition. We will come back on the qualifiers in Section 2.4.4.10.

A class must belong to one schema, and its name must be unique within that particular schema. If some new management objects must be defined as new classes in CIM, a developer can use the CIMOM API or a MOF file to define its particularities. We will revisit MOF files in Section 2.5.

2.4.4.3 Superclasses and subclasses

In WBEM, subclasses can also be referred to as specialization classes. In the same way, a superclass, which is a parent class, can also be referred to as a generalization class. As we have seen before, the class definition always starts from the most general representation of the managed object. The organization of classes is like an upside-down tree with a base class at the root. The superclasses and the subclasses illustrate the notion of inheritance. It is important to note that multiple inheritances are not supported. This means that each class inherits from exactly one superclass. In the cheeseburger example, the *meat* class is a base class. The *beef* class and the *pork* class inherit only from the *meat* class. In the same way, in a computer system, we see in Figure 2.9 that the *CIM_LogicalDevice* class is a subclass of the *CIM_LogicalElement* class.

2.4.4.4 Domain and range

Properties and methods are declared within the scope of a class. The declaring class is referred to as the domain for the property or method. If you think about the ingredients used to produce a cheeseburger (see Figure 2.3), we have the properties, such as the *name* property, and a method, which is the *IsGoodToEat* method. The scope of the property and the method is limited to the class of the ingredient. In our example, the scope of the *name* property and the *IsGoodToEat* method is limited to the *cheese*, the *meat*, the *sauce*, the *vegetable*, and the *bread* classes.

2.4.4.5 Properties

Properties are values representing characteristics of a class. A property has a name and only one domain, which is the class that owns the property. In the case of the classes representing the cheeseburger ingredients, each ingredient has a name (with a value coded as a string), a production date (with a value coded as a datetime string), and a validity period (with a value coded

as unsigned integer of 8 bits). As mentioned before, properties can also be inherited from superclasses by subclasses.

2.4.4.6 Methods

A method is an operation describing the behavior of an instance of a class. Not all classes have methods. A method has a name and only one domain, which is the class that owns the method. In the case of the classes representing the cheeseburger ingredients, each ingredient has a method called is *IsGoodToEat*. This method returns a Boolean value True or False to determine the freshness of the ingredient. Like properties, subclasses can inherit methods from superclasses.

2.4.4.7 Associations

An association is a class that contains two or more references. In the example of the cheeseburger, we have five references: *MeatIngredient*, *CheeseIngredient*, *SauceIngredient*, *VegetableIngredient*, and *BreadIngredient*. Each of these references points to the ingredients that are classes. References are properties of associations. An association represents a relationship between two or more objects. An association is a class type with a qualifier stating the nature of the class (an association qualifier). For instance, in Figure 2.16 (right pane), the *CIM_LogicalDiskBasedOnPartition* class is an association, as it associates the *CIM_DiskPartition* and *CIM_LogicalDisk* classes.

2.4.4.8 References

Talking about associations always implies the notion of references. A reference defines the role each object plays and represents the role name of a class in the context of an association. Therefore, the domain of a reference is an association. In the cheeseburger example, the reference between the cheeseburger and each ingredient determines the role of the ingredient, which constitutes the cheeseburger (see Figure 2.7). Associations can support multiple relationships for a given object, which implies that we may have many references. In computing words, a system can be related to many system components in many different ways. For instance, in Figure 2.16 (right pane of **WMI CIM Studio**), the references are illustrated by two arrows starting from *CIM_LogicalDiskBasedOnPartition* and pointing to the *CIM_DiskPartition* and *CIM_LogicalDisk* classes.

2.4.4.9 Indications

An indication is an object created as a result of the occurrence of an event. Indications are always subclasses and may contain properties or methods.

Since an indication is the result of an event, such a class type may have an association with one or more triggers. Triggers create the instance of the indications. In Chapter 6, we will examine what events are and how to work with them. For now, remember that an indication is nothing more than an instance coming from an event.

2.4.4.10 Qualifiers

A qualifier is used to characterize classes, properties, methods, and other elements of the CIM schema. When we talked about the associations, we saw that an association is a class of a particular type. To determine that a given class is an association class, a qualifier is used. Qualifiers are also used to determine whether a property of a class is required and to define the key of a class. Keys are properties that provide a unique value to distinguish instances of a class by using one or more properties as a unique identifier. In such a case, we refer to a key qualifier. Note that a singleton class (a class that has only one instance) does not need a key because it is already unique.

A qualifier has a name, a type, a value, a scope, and a flavor. Flavors describe rules that determine whether a qualifier can be propagated to a derived class or instance. It also determines whether a derived class or instance can override the qualifier's original value. In CIM, a qualifier may have a default value. The type of the default value must agree with the qualifier type (association qualifier, property qualifier, key qualifier, etc.). Figure 2.17 shows a logical representation of a qualifier with its miscellaneous possible values. Basically, qualifiers are sorted into four categories:

- **Metaqualifiers:** This qualifier type is used to define the actual usage of a class or property definition.
- **Standard qualifiers:** These qualifiers are CIM-compliant qualifiers. They are implemented by any compliant CIM product.
- **Optional qualifiers:** Optional qualifiers address situations that are uncommon to all CIM-compliant implementations. They are provided in the CIM specification to avoid random user-defined qualifiers.
- **WMI-specific qualifiers:** These qualifiers are especially added by Microsoft to support the WMI implementation. They are not part of the CIM specification and are purely Microsoft-related.

Qualifiers are visible in **WMI CIM Studio** by right-clicking on a class or an instance, or on a property or method of a class. Table 2.2 contains the lists of qualifiers available.

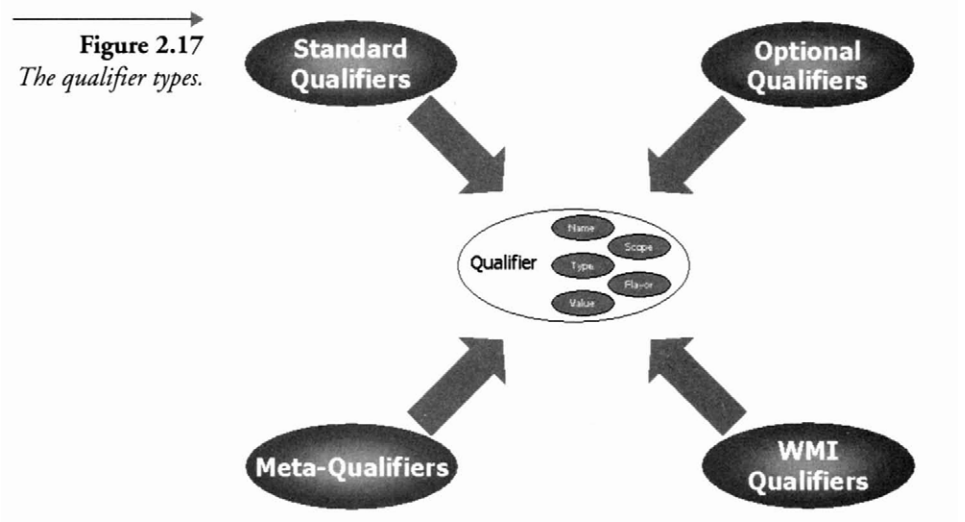

Figure 2.17
The qualifier types.

To avoid repetition of the Microsoft Platform SDK, as the list of qualifier is quite long, you can refer to WMI SDK to get the complete meaning of the available qualifiers.

Table 2.2 *Available Qualifiers*

Standard Qualifiers	Optional Qualifiers	MetaQualifiers	WMI Qualifiers
Abstract	Delete	Association	Amendment
Aggregate	Expensive	Indication	CIM_Key
Aggregation	IfDeleted		CIMType
Alias	Indexed		ClassContext
ArrayType	Invisible		Deprecated
Bitmap	Large		Display
Bitvalues	Not_Null		Dynamic
Description	Provider		DynProps
DisplayName	Syntax		ID
Gauge	SyntaxType		Implemented
In	TriggerType		InstanceContext
In, Out	UnknownValues		Locale
Key	UnsupportedValues		Optional
MappingStrings			Privileges
Max			PropertyContext

Table 2.2 *Available Qualifiers (continued)*

Standard Qualifiers	Optional Qualifiers	MetaQualifiers	WMI Qualifiers
MaxLen			Provider
MaxValue			Singleton
Min			Static
MinValue			SubType
ModelCorrespondence			UUID
Nonlocal			WritePrivileges
NonlocalType			
NullValue			
Out			
Override			
OverrideValue			
Propagated			
Read			
Required			
Revision			

2.4.4.11 Override

We have seen that classes are organized similarly to an upside-down tree. Each class has properties and methods. WBEM introduces the override relationship to indicate the substitution between a property or a method of a subclass and a property or a method inherited from a superclass. By using the override relationship, it is possible to overwrite a property or a method in a subclass from a superclass. This relation is made with the help of qualifiers and is illustrated in Figure 2.13 with the dashed line.

2.5 What is a MOF file?

A MOF file is an ASCII file using a specific syntax and language definition to code CIM definitions. MOF files are used to load the CIM repository with new definitions of managed objects without having to use the specific API to modify or extend the CIM Schema. In some ways, MOF files can be compared to the LDIF files used in the LDAP world. MOF files can be used to export or import a set of information to or from the CIM repository, just as LDIF files can be used to export or import a set of information

from an LDAP directory. MOF files encode the definition of CIM objects as stated in the CIM specification. For example, you can implement the cheeseburger example in a MOF file. This produces the MOF file shown in Sample 2.1. With the knowledge we have of the cheeseburger data model we developed in UML notation, it will be quite easy to understand the MOF file content. Note that this MOF file creates a new namespace in the CIM repository. As the principle is the same for all ingredients, we skipped some lines representing ingredients in the reproduction of this MOF file.

Sample 2.1 *The cheeseburger MOF file*

```
1:// File: Food.MOF
2:
3:// Changes focus to Root namespace
4:#pragma namespace("\\\\.\\Root")
5:
6:// Create new namespace
7:instance of __Namespace
8:{
9:    Name = "Food";
10:};
11:
12:// Changes focus to new namespace
13:#pragma namespace("\\\\.\\ROOT\\Food")
14:
15:// class: Bread
16:// Derived from:
17:[abstract: ToInstance ToSubclass]
18:class Bread
19:{
20:        [Description ("This is the bread name"): ToInstance ToSubclass,
           read: ToInstance ToSubclass]
21:        string Name;
22:        [Description ("This is the bread production date"): ToInstance ToSubclass,
           read: ToInstance ToSubclass]
23:        datetime ProductionDate;
24:        [Description ("This is the bread validity period"): ToInstance ToSubclass,
           read: ToInstance ToSubclass]
25:        datetime ValidDays;
26:        [Description ("Is the bread still good to eat?"): ToInstance ToSubclass]
27:        boolean IsGoodToEat();
28:};
29:
30:// class: ToastedBun
31:// Derived from: Bread
32:class ToastedBun : Bread
33:{
34:        [Key : ToInstance ToSubclass,
           Description ("This is the toasted bun name"): ToInstance ToSubclass]
35:        string Name;
36:};
37:
38:// class: Meat
39:// Derived from:
40:[abstract: ToInstance ToSubclass]
41:class Meat
42:{
```

```
43:         [Description ("This is the meat name"): ToSubclass,
             read: ToInstance ToSubclass]
44:         string Name;
45:         [Description ("This is the meat production date"): ToSubclass,
             read: ToInstance ToSubclass]
46:         datetime ProductionDate;
47:         [Description ("This is the meat validity period"): ToSubclass,
             read: ToInstance ToSubclass]
48:         datetime ValidDays;
49:         [Description ("Is the meat still good to eat?"): ToSubclass]
50:         boolean IsGoodToEat();
51:};
..:
61:// class: Beef
62:// Derived from: Meat
63:class Beef : Meat
64:{
65:         [Key : ToInstance ToSubclass,
             Description ("This is the beef name"): ToInstance ToSubclass]
66:         string Name;
67:};
..:
77:// class: Vegetable
78:// Derived from:
79:[abstract: ToInstance ToSubclass]
80:class Vegetable
81:{
82:         [Description ("This is the vegetable name"): ToSubclass,
             read: ToInstance ToSubclass]
83:         string Name;
84:         [Description ("This is the vegetable production date"): ToSubclass,
             read: ToInstance ToSubclass]
85:         datetime ProductionDate;
86:         [Description ("This is the vegetable validity period"): ToSubclass,
             read: ToInstance ToSubclass]
87:         datetime ValidDays;
88:         [Description ("Is the vegetable still good to eat?"): ToSubclass]
89:         boolean IsGoodToEat();
90:};
91:
92:// class: Salad
93:// Derived from: Vegetable
94:class Salad : Vegetable
95:{
96:         [Key : ToInstance ToSubclass,
             Description ("This is the salad name"): ToInstance ToSubclass]
97:         string Name;
98:};
99:
100:// class: Gherkin
101:// Derived from: Vegetable
102:class Gherkin : Vegetable
103:{
104:         [Key : ToInstance ToSubclass,
             Description ("This is the gherkin name"): ToInstance ToSubclass]
105:         string Name;
106:};
...:
116:// class: Potatoes
117:// Derived from: Vegetable
118:class Potatoes : Vegetable
```

```
119:{
120:        [Key : ToInstance ToSubclass,
             Description ("This is the potatoes name"): ToInstance ToSubclass]
121:        string Name;
122:};
123:
124:// class: FrenchFries
125:// Derived from: Potatoes
126:class FrenchFries : Potatoes
127:{
128:        [Description ("This is the french fries name"): ToInstance ToSubclass]
129:        string Name;
130:        [Description ("This is the french fries size"): ToInstance ToSubclass,
             ValueMap{"Small", "Medium", "Large"}: ToInstance ToSubclass,
             read: ToInstance ToSubclass]
131:        string Size;
132:};
133:
134:// class: Cheese
135:// Derived from:
136:[abstract: ToInstance ToSubclass]
137:class Cheese
138:{
139:        [Description ("This is the cheese name"): ToSubclass, read: ToInstance ToSubclass]
140:        string Name;
141:        [Description ("This is the cheese production date"): ToSubclass,
             read: ToInstance ToSubclass]
142:        datetime ProductionDate;
143:        [Description ("This is the cheese validity period"): ToSubclass,
             read: ToInstance ToSubclass]
144:        datetime ValidDays;
145:        [Description ("Is the cheese still good to eat?"): ToSubclass]
146:        boolean IsGoodToEat();
147:};
...:
165:// class: ProcessedCheeseSlice
166:// Derived from: Cheese
167:class ProcessedCheeseSlice : Cheese
168:{
169:        [Key : ToInstance ToSubclass,
             Description ("This is the processed cheese slice name"): ToInstance ToSubclass]
170:        string Name;
171:};
172:
173:// class: Sauce
174:// Derived from:
175:[abstract: ToInstance ToSubclass]
176:class Sauce
177:{
178:        [Description ("This is the sauce name"): ToSubclass,
             read: ToInstance ToSubclass]
179:        string Name;
180:        [Description ("This is the sauce production date"): ToSubclass,
             read: ToInstance ToSubclass]
181:        datetime ProductionDate;
182:        [Description ("This is the sauce validity period"): ToSubclass,
             read: ToInstance ToSubclass]
183:        datetime ValidDays;
184:        [Description ("Is the sauce still good to eat?"): ToSubclass]
185:        boolean IsGoodToEat();
186:};
```

```
187:
188:// class: Ketchup
189:// Derived from: Sauce
190:class Ketchup : Sauce
191:{
192:        [Key : ToInstance ToSubclass,
             Description ("This is the ketchup name"): ToInstance ToSubclass]
193:        string Name;
194:};
...:
212:// class: CheeseBurger
213:// Derived from:
214:[abstract: ToInstance ToSubclass]
215:class CheeseBurger
216:{
217:        uint32 Weight;
218:};
219:
220:// class: Beverage
221:// Derived from:
222:class Beverage
223:{
224:        [Key : ToInstance ToSubclass,
             Description ("This is the beverage name"): ToInstance ToSubclass]
225:        string Name;
226:};
227:
228:// class: Coke
229:// Derived from: Beverage
230:class Coke : Beverage
231:{
232:        [Description ("This is the Coke name"): ToInstance ToSubclass,
             read: ToInstance ToSubclass]
233:        string Name;
234:        [Description ("This is the Coke size"): ToInstance ToSubclass,
             ValueMap{"Small", "Medium", "Large"}: ToInstance ToSubclass,
             read: ToInstance ToSubclass]
235:        string Size;
236:};
237:
238:// class: MenuBurgerAggregation
239:// Derived from:
240:[association: ToInstance ToSubclass]
241:class MenuBurgerAggregation
242:{
243:        BurgerMenu ref Menu;
244:        CheeseBurger ref Burger;
245:};
246:
247:// class: Cheeseassociation
248:// Derived from:
249:[association: ToInstance ToSubclass]
250:class Cheeseassociation
251:{
252:        ProcessedCheeseSlice ref CheeseIngredient;
253:        CheeseBurger ref Burger;
254:};
255:
256:// class: Meatassociation
257:// Derived from:
```

```
258:[association: ToInstance ToSubclass]
259:class Meatassociation
260:{
261:        CheeseBurger ref Burger;
262:        Beef ref MeatIngredient;
263:};
264:
265:// class: Vegetableassociation
266:// Derived from:
267:[association: ToInstance ToSubclass]
268:class Vegetableassociation
269:{
270:        CheeseBurger ref Burger;
271:        Gherkin ref VegetableIngredient;
272:};
273:
274:// class: Sauceassociation
275:// Derived from:
276:[association: ToInstance ToSubclass]
277:class Sauceassociation
278:{
279:        CheeseBurger ref Burger;
280:        Ketchup ref SauceIngredient;
281:};
282:
283:// class: Breadassociation
284:// Derived from:
285:[association: ToInstance ToSubclass]
286:class Breadassociation
287:{
288:        ToastedBun ref BreadIngredient;
289:        CheeseBurger ref Burger;
290:};
291:
292:// class: BurgerMenu
293:// Derived from:
294:class BurgerMenu
295:{
296:};
297:
298:// class: MenuBeverageAggregation
299:// Derived from:
300:[association: ToInstance ToSubclass]
301:class MenuBeverageAggregation
302:{
303:        BurgerMenu ref Menu;
304:        Coke ref Beverage;
305:};
306:
307:// class: MenuFriesAggregation
308:// Derived from:
309:[association: ToInstance ToSubclass]
310:class MenuFriesAggregation
311:{
312:        BurgerMenu ref Menu;
313:        FrenchFries ref FrenchFries;
314:};
315:
316:// EOF Food.MOF
```

To load this MOF file in the CIM repository, it is necessary to use the MOF compiler, called **MOFCOMP.Exe**. We see how to use the MOF compiler that comes with WMI in Section 2.8.5.1. Basically, the command line to use in this particular case is as follows:

```
C:\>MOFCOMP.Exe Food.Mof
```

Once loaded in the CIM repository, this MOF file produces the output in Figure 2.18. We have set the **WMI CIM Studio** view on the cheeseburger class created with the different ingredients to show the result of the associations.

Microsoft provides a MOF editor on the DMTF Web site. You can download it for free from http://www.dmtf.org/standards/cimtools.php. The MOF editor eases the reading of MOF files, as the ASCII representation can be sometimes quite confusing to read.

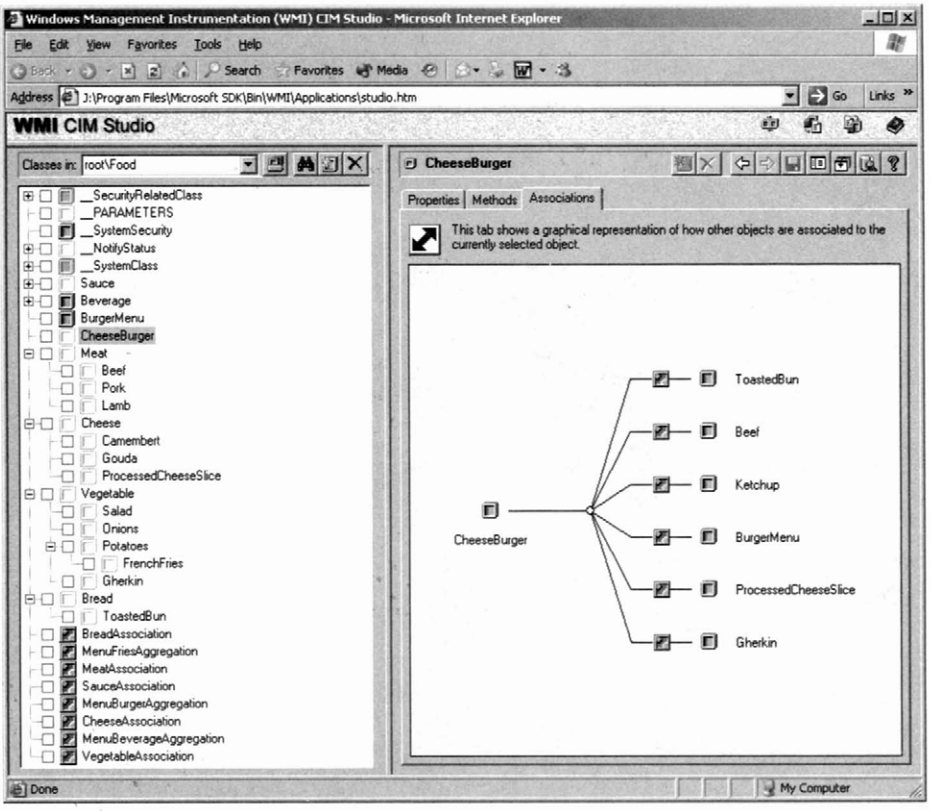

Figure 2.18 *The cheeseburger association class once created with a MOF file.*

Although it is important to have a basic understanding of the MOF file representation, a deep knowledge of MOF file creation is not required at all to script with WMI. However, if you are a WMI developer, it will be mandatory for you to dive into the MOF syntax details since a MOF file is required to register your WMI providers and their associated classes. Programming of WMI providers is beyond the scope of this book, but if you want to find out more about MOF files, we recommend that you take a look at the WMI SDK.

2.6 What is xmlCIM?

A MOF file defines a textual declaration format, but it is not a standard for CIM information interchange between systems. In order to be able to communicate management information between different systems, two aspects must be taken into consideration: The first consideration is to use a communication protocol common to any management platform; the second is to use a common representation of the managed information for the transport between platforms. The CIM specification defines HTTP as the common communication protocol. This makes sense because HTTP is available on every computer system in the industry. As a common representation, XML is used with a Document Type Definition (DTD). This also makes sense, because XML is understandable by any platform, and parsing components are widely available in the industry.

There are many different ways in which CIM operations can be represented within XML with the operations encapsulated in HTTP messages. For interoperability reasons between different implementations of CIM, there is an obvious requirement for standardization of both the XML representation of CIM and the HTTP encapsulation. For this reason xmlCIM defines the declarations of the CIM objects that can be transformed in XML and the CIM messages for encapsulation over HTTP. Under Windows.NET Server, it is possible to obtain the XML representation of the CIM objects, but it is not yet possible to communicate over HTTP. If you need more information about this subject, you can visit the DMTF Web site at http://www.dmtf.org/standards/published_documents.php.

2.7 The supported platforms

For Microsoft platforms, WMI is the Microsoft implementation of WBEM. It is installed by default under Windows.NET Server, Windows 2000, and Windows Millennium. It is possible to install WMI on platforms

such as Windows 95, Windows 98, and Windows NT 4.0 (from SP4 or higher) as a separate download from the Microsoft download center at http://www.microsoft.com/downloads/search.asp. Since the release of Windows 2000, the Windows Driver Model also supports WMI, which means that a driver developer can expose some interesting information to WMI. This is supported only under Windows.NET Server, Windows XP, Windows 2000, Windows 98, and Windows Millennium. As mentioned before, WMI runs as a service under Windows NT, Windows 2000, and Windows.NET Server, whereas it runs as an application under Windows 98, Windows 95, and Windows Millennium. This is due to the structural differences between Windows NT and Windows 9x/Me. The WMI SDK is supported only on Windows.NET Server, Windows 2000, and Windows NT 4.0 (sP4 or higher).

In addition to the Microsoft WBEM implementation, HP has implemented WBEM on its HPUX and Tru64 UNIX platforms. The software component is called HP WBEM Services for HPUX (or Tru64). You can refer to http://www.hp.com/large/infrastructure/management/wbem for more information. HP also offers the HP WBEM Services for HP-UX SDK. This software component for HPUX platforms is the developer's kit for the HP WBEM Services for HP-UX product. This product is based on The Open Group (TOG) Pegasus Open Source Software (OSS) project (http://www.opengroup.org/pegasus).

SUN also announced its implementation of WBEM on the Solaris 8 Operating System. The official name is "Solaris™ WBEM Services and Sun™ WBEM SDK Support for XML and Java 2™ Technology." You can find the public announcement at http://www.sun.com/software/solaris/wbem/cover and additional details at http://www.sun.com/software/solaris/wbem/faq.html. The software kit is available for download at the same location. You can also contact SUN at wbem-support@sun.com for more information.

Novell also started the Zero Effort Network (ZEN) initiative. You can check the following URL for more information: http://www.novell.com/news/leadstories/98/jul22. Unfortunately, at the time of this writing, Novell does not yet implement the CIM database in its ZENworks for Desktops 3 Inventory Database (ZfD3). However, as part of the ZENworks for Networks, Novell also included some DMTF standards, such as DMI, for hardware inventory scanning in ZfD3 (see Section 2.2). ZfD3 is not compatible with WBEM, but desktop side clients could be compatible with both DMI and WBEM. Today, ZfD3 requires the DMI layer functionality in order to

retrieve full hardware inventory correctly. See http://www.novell.com for more information.

Among the Novell, SUN, and Microsoft initiatives, there is also the WBEMsource initiative. The WBEMsource initiative is an umbrella organization providing coordination between open-source WBEM projects. Its goal is to achieve interoperability and portability. Individual development projects are completely autonomous projects that are voluntarily participating in the initiative. The mission of the WBEMsource initiative is to promote the widespread use of the DMTF's management technologies through the creation of an environment fostering open-source WBEM implementations. You can refer to http://www.opengroup.org/wbemsource for more information.

Although this book focuses on the Microsoft implementation of WBEM called WMI, it is quite clear that the WBEM implementation is not a Microsoft-only initiative.

2.8 The Microsoft CIM implementation: WMI

WMI is designed as a Schema extension of the CIM Core Schema and the CIM Common Model. WMI is implemented for the Windows platforms. WMI enables Windows systems, applications, networks, and other managed components to be represented using CIM as designed by the DMTF.

In addition to data modeling, WMI offers a set of services that includes query-based information retrieval and event notification. The Component Object Model (COM) programming interface allows access to the services and the management data while implementing an abstraction layer.

2.8.1 WMI architecture

WMI is implemented as a three-tiered architecture (see Figure 2.19). At the top of the architecture, we have the management applications. These applications interact with the WMI COM API. The WMI COM API has a collection of COM objects that are usable from various programming languages such as Visual Basic, C++, and scripting languages such as JScript or VBScript. Behind the WMI COM API, we find the Common Information Model Object Manager (CIMOM) component. CIMOM acts as a broker between the management applications (called the consumers), the CIM repository and the WMI providers.

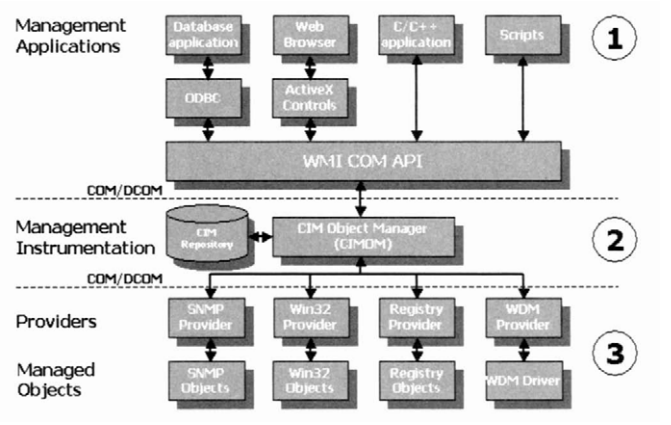

Figure 2.19 *The WMI architecture.*

2.8.2 Providers and consumers

The WMI architecture introduces the notion of providers and consumers. Providers are the components accessing the management objects to gather the management data (layer 3 in Figure 2.19). On the other hand, a consumer is an application that uses the data collected by the providers (layer 1 in Figure 2.19). Basically, consumers will be applications such as HP OpenView Operations for Windows, Microsoft Operations Manager (MOM), or any other management application that consumes WMI information. The scripts developed in this book are also WMI Consumers. This layered architecture implies that the CIM repository has interfaces with an (upper) API for management applications (consumers) and an (lower) API for providers.

WMI providers are installed with the Windows system installation or with any installed application that provides its own WMI providers. This is why there is a distinction between the built-in WMI providers and application-specific WMI providers. For example, when Exchange 2000 is installed, it adds the necessary definitions (classes) related to the Exchange WMI providers to the CIM repository. The WMI architecture is extensible by simply creating new providers and adding the new classes to the CIM repository; as a result we have an unlimited list of available WMI providers. Table 2.3 shows a list of the most important WMI providers delivered with Windows.NET Server.

Application developers can add their own providers and extract the management data from their applications. By using classes, properties, and methods, developers can represent the management data in the CIM data model. To summarize, the provider is an abstraction layer for CIMOM, which hides the access complexity and interpretation of the management

Table 2.3 *Some Windows.NET Server WMI Providers*

Provider	Description
Directory Services Provider	Makes the classes and objects in the Microsoft® Active Directory™ available to WMI management applications
Event Log Provider	Provides access to data and notifications of events from the Microsoft® Windows_NT®/Microsoft® Windows®_2000 Event Log
Microsoft Windows Installer Provider	Provides access to information about applications that are installed using Windows Installer
Performance Counters Provider	Provides access to raw performance counter data
Performance Monitor Provider	Provides access to data from the Windows_NT/Windows_2000 Performance Monitor
Power Management Event Provider	Represents power management events resulting from power state changes
Registry Event Provider	Sends an event whenever a change occurs to a key, a value, or an entire tree in the registry
Registry Provider	Provides access to data from the registry
Security Provider	Provides access to security settings that control ownership, auditing, and access rights to the NTFS file system
SNMP Provider	Provides access to data and events from SNMP devices
View Provider	Creates new classes made of properties from different source classes, namespaces, or computers
WDM Provider	Provides access to data and events from device drivers that conform to the WMI interface
Win32 Provider	Provides access to data from the Win32 subsystem

data of a specific implementation. Once completed, consumers can access the class definitions related to the applications and retrieve the management data through CIMOM and the especially developed provider.

More than being a simple "pass-through" to access the management information, providers can be event driven. Simply said, an application can register with CIMOM to receive notifications about a particular event. In this case, CIMOM works in conjunction with the WMI providers to be notified about the expected events. Once the event occurs, the WMI provider notifies CIMOM. Next, CIMOM notifies the consumers that have made a registration for the event. This feature is a very important and useful function because it leaves an application or a script in a dormant state until the expected event occurs. WMI watches the event on behalf of the consumer. CIMOM is an abstraction layer for the consumers of the management information provided by the WMI providers. This abstraction layer allows the correlation of events coming from various WMI providers.

WMI providers are .dll files located in %SystemRoot%\system32\wbem. If you examine this directory, you will see a collection of .dll files. In addition to these DLLs, there is typically one file with the same name that uses a .mof extension. The MOF file is the file containing the necessary definitions to register the WMI provider and its related set of classes created in the CIM repository.

Beside the providers installed in a Windows system, WMI also provides some consumers. This means that consumers include not only the applications or scripts developed on top of WMI but also the components brought by WMI. Actually, WMI comes with a set of ready-to-use consumers that perform actions based on specific management events. You can associate a particular event monitored by a WMI provider and redirect this event to a built-in WMI consumer to perform a particular task. In this case, there is nothing to develop; only a registration of the expected event with the associated WMI consumer is required. Usually, the registration with CIMOM is made with the help of a MOF file provided with the consumer. Some of the WMI consumers provided with Windows.NET Server are listed in Table 2.4.

There is a clear distinction between two types of WMI consumers. There are permanent consumers and temporary consumers. Briefly, a permanent

Table 2.4 *Some Windows.NET Server WMI Consumer Providers*

Consumer	Description
SMTP Event Consumer	Sends an e-mail message using SMTP each time an event is delivered to it.
WMI Event Viewer	A permanent event consumer that allows users to sort and view the details of events received. As events are generated in WMI by Windows Management or by event providers, event objects are forwarded to any consumers registered for these types of events. You can register WMI Event Viewer for any event filters, so you can view only incoming events that match the filters.
Active Script Event Consumer	Executes a predefined script in an arbitrary scripting language whenever an event is delivered to it.
Command Line Event Consumer	Launches an arbitrary process in the local system context whenever an event is delivered to it.
Log File Event Consumer	Writes customized strings to a text log file whenever events are delivered to it.
NT EventLog Event Consumer	Logs a specific message to the Windows_NT event log whenever an event is delivered to it.

consumer receives event notifications regardless of whether it is running. The registration of the consumer is made in the CIM repository with a system class. As the registration is permanent in the CIM repository, WMI locates the consumer and starts it if necessary. A permanent consumer is written as a COM object. On the other hand, a temporary consumer is a component that makes a registration to receive notifications of events only when it is active. Once the component (an application or a script) is no longer running, notifications are no longer received. WMI temporary consumers will be used when we discuss scripting with WMI events in Chapter 6.

2.8.3 The CIM repository

The CIM repository is the database used by WMI to store the CIM data model. It is also called in the Microsoft documentation the WMI repository. In a Windows system, the CIM repository is located in \%SystemRoot%\System32\WBEM\Repository folder. No application directly accesses the CIM repository. Every access to the CIM repository goes through CIMOM. This is the only way that management applications can access the CIM repository (through the WMI COM API). The same rule applies for the WMI providers (by interacting with CIMOM).

By default, the CIM repository is saved in a .rec file every 30 minutes. Most of the default settings of WMI can be controlled with the computer management MMC as shown in Figure 2.20.

The CIM repository is initialized from various MOF files (located in %SystemRoot%\System32\wbem) at WMI installation time. If the CIM repository is damaged or deleted, when WMI is restarted the CIM repository is reinitialized from a collection of MOF files listed in a registry key named "Autorecover MOFs" and located in HKLM\SOFTWARE\Microsoft\WBEM\CIMOM. Only the MOF files listed in this key are used to reinitialize the CIM repository. If any other installed applications extend the CIM repository for their own purposes, make sure that the MOF files used are also listed in this registry key. You can manually recompile the required MOF file later on with the MOF compiler (**MOFCOMP.Exe**).

Under Windows XP and Windows.NET Server, the CIM repository resides in the %SystemRoot%\system32\wbem\Repository\FS folder. This folders contains the following files:

- **Index.btr**: Binary-tree (btree) index file
- **Index.map**: Transaction control file

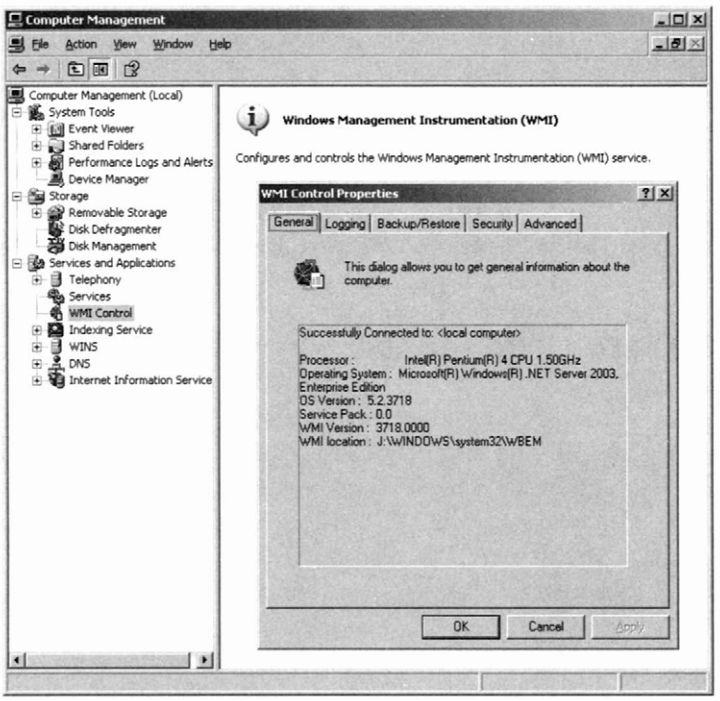

Figure 2.20 *The CIM repository automatic backup.*

- **Objects.data:** CIM repository where managed resource definitions are stored
- **Objects.map:** Transaction control file

Under Windows 2000 and Windows NT 4.0 Service Pack 4, the CIM repository is located in the %SystemRoot%\system32\wbem\Respository folder. This folder contains **cim.rep** file, which implements the CIM database. Under Windows Millennium Edition (Me), Windows 98, and Windows 95, the **cim.rep** file is also the CIM database file. However, it is located in the %windir%\system\wbem\Respository folder.

2.8.4 The WMI query language

The WMI query language (WQL) is a subset of the ANSI SQL language with minor semantic changes to suit the WMI needs. WQL is a retrieval language only. It does not allow operations to modify, insert, or delete information. WQL is not part of the DMTF standard, but it has been proposed to be part of the standard. At the time of writing, WQL is under examination to be included in the DMTF standard. This language is mainly used to perform:

- **Data queries:** These queries are used to retrieve instances of classes. For example, to retrieve the instances of the *Win32_Service* class:

```
Select * From 'Win32_Service'
```

- **Event queries:** These queries are used by temporary consumers, permanent consumers, and event providers. Basically, event queries are used to filter events. For example, to receive a notification when a change is made to one *Win32_Service* instance (stopping or starting a service, for example):

```
Select * From __InstanceModificationEvent Within 10 Where
TargetInstance ISA 'Win32_Service'
```

- **Schema queries:** These queries are used to retrieve class definitions and information about schema associations. For example, to retrieve the class definition of the *Win32_Service* class:

```
SELECT * FROM meta_class Where __this ISA 'Win32_Service'
```

Or to retrieve the associations of the *Win32_Service* SNMP instance:

```
ASSOCIATORS OF {Win32_Service.Name='SNMP'}
```

When practicing exercises with the WMI Microsoft tools, we use two basic queries: one data query to retrieve the list of the Windows services and one event query to receive event notification changes performed to one or more Windows services. We revisit the possibilities offered by WQL in the following chapter.

2.8.5 The WMI Microsoft tools

As part of the WMI standard installation and the Platform SDK, Microsoft provides some tools to ease working with WMI. It is important to be familiar with these tools, because they will be helpful to discover, extend, and troubleshoot WMI.

2.8.5.1 The MOF compiler: MOFCOMP.exe

The MOF compiler parses a file containing MOF statements and adds the classes and class instances defined in the file to the CIM repository. **MOFCOMP.exe** is included in every WMI installation. Every definition existing in the CIM repository is initially defined in a MOF file. MOF files are located in %SystemRoot%\system32\wbem. During the WMI setup, they are loaded in the CIM repository.

Sample 2.1 showed a MOF file example (**Food.mof**) for the cheeseburger representation. The following example shows the MOF file (**SMTP-**

Cons.mof) used to register the permanent *SMTP* event consumer, SMTPCons.dll. This permanent consumer is a registered COM object DLL using the class ID C7A3A54B-0250-11d3-9CD1-00105A1F4801.

```
 1:// Copyright (c) 1997-2001 Microsoft Corporation, All Rights Reserved
 2:
 3:[locale(1033)]
 4:class SMTPEventConsumer : __EventConsumer
 5:{
 6:  [key] string Name;
 7:  [not_null, write] string SMTPServer;
 8:  [Template, write] string Subject;
 9:  [Template, write] string FromLine;
10:  [Template, write] string ReplyToLine;
11:  [Template, write] string Message;
12:  [Template, write] string ToLine;
13:  [Template, write] string CcLine;
14:  [Template, write] string BccLine;
15:  [write] string HeaderFields[];
16:};
17:
18:Instance of __Win32provider as $SMTPCONS_P
19:{
20:  Name = "SMTPEventConsumer";
21:  Clsid = "{C7A3A54B-0250-11d3-9CD1-00105A1F4801}";
22:  HostingModel = "LocalSystemHost";
23:
24:};
25:
26:Instance of __EventConsumerproviderRegistration
27:{
28:  provider = $SMTPCONS_P;
29:  ConsumerclassNames = {"SMTPEventConsumer"};
30:};
..:
..:
..:
```

Briefly, we see that the first section defines the *SMTPConsumer* class derived from the *__EventConsumer* system class (line 4). This means that the *SMTPConsumer* class inherits from *__EventConsumer* system class. Next, the section contains the properties exposed by the consumer (lines 5 to 16). Notice that the first property of the consumer is its name (line 6) and is defined as a key. This means that the name property is used to uniquely identify the consumer. This implies that the name is a mandatory property to create an instance of the consumer. Next, there are two additional sections:

1. One section references the consumer provider DLL file (lines 18 to 24), which is an instance of the *__Win32provider* system class used to register the WMI providers.

2. The next section registers the component as a consumer provider in the CIM repository (lines 26 to 30). This section creates the link between the *SMTPConsumer* class definition (lines 5 to 43) and the provider itself (lines 18 to 24).

To load this MOF file in the CIM repository, the MOF compiler is used as follows from the command line:

```
C:\>MOFCOMP %SystemRoot%\System32\wbem\SMTPCons.MOF
```

We will not enter in the security considerations now (as this is deeply examined in the second volume dedicated to WMI, *Leveraging Windows Management Instrumentation Scripting*, ISBN 1555582990) but be aware that to register a permanent event consumer, you must be a member of the **Administrators** group.

By default every MOF file is loaded in the **Root\Default** namespace. Use the –N switch to overwrite the default. To get a complete list of the options available from **MOFCOMP.exe**, refer to Table 2.5.

Table 2.5 *MOFCOMP.exe Switches*

MOF Compiler Switches	
-autorecover	Adds the named MOF file to the list of files compiled during repository recovery. The list of autorecover MOF files is stored in the registry key named: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\Autorecover MOFs. The MOF files listed in this registry entry must reside on the local computer because MOF files that use the autorecover command cannot recover MOF files located on a remote computer.
-check	Requests that the compiler perform a syntax check only and print appropriate error messages. No other switch can be used with this switch. When this switch is used, no connection to WMI is established and no modifications to the CIM repository are made.
-N:<*namespacepath*>	Requests that the compiler load the MOF file into the namespace specified as "namespacepath." The default namespace (root\default) is used unless this switch is used or a #pragma namespace ("namespace path") statement appears in the MOF file. If both the -N namespace switch and #pragma namespace are used, #pragma namespace takes priority.
-class:createonly	Requests that the compiler not make any changes to existing classes. When this switch is used, the compilation terminates if a class specified in the MOF file already exists.
-class:forceupdate	Forces updates of classes when conflicting child classes exist. For example, suppose a class qualifier is defined in a child class and the base class tries to add the same qualifier. In -class:forceupdate mode, the MOF compiler resolves this conflict by deleting the conflicting qualifier in the child class. If the child class has instances, the forced update fails.

Table 2.5 *MOFCOMP.exe Switches (continued)*

MOF Compiler Switches	
-class:safeupdate	Allows updates of classes even if there are child classes, as long as the change does not cause conflicts with child classes. For example, this flag allows adding a new property to the base class that was not previously mentioned in child classes. If the child classes have instances, the update fails.
-class:updateonly	Requests that the compiler not create any new classes. When this switch is used, the compilation terminates if a class specified in the MOF file does not exist.
-instance:updateonly	Requests that the compiler not create any new instances. When this switch is used, the compilation terminates if an instance specified in the MOF file does not exist.
-instance:createonly	Requests that the compiler not make any changes to existing instances. When this switch is used, the compilation terminates if an instance specified in the MOF file already exists.
-B:<*filename*>	Requests that the compiler create a binary version of the MOF file with the name "filename" without making any modifications to the CIM repository.
-WMI	Requests that the compiler perform a WMI syntax check. The -B switch must be used with this switch. The -WMI switch is only used for building binary MOFs for use by WDM device drivers. This switch invokes a separate binary MOF file checker, which runs after the binary MOF file is created. To use this switch under Windows 2000, the Windows 2000 DDK must first be installed.
-P:<Password>	Specifies Password as the password for the computer's user to enter when logging on.
-U:<UserName>	Specifies UserName as the name of the user logging on.
-A:<Authority>	Specifies Authority as the authority (domain name) to use when logging on to WMI.
-MOF:<*path*>	Name of the language-neutral output. Used with the -AMENDEMENT switch to specify the name of the language-neutral MOF file that will be generated.
-MFL:<*path*>	Name of the language-specific output. Used with the -AMENDEMENT switch to specify the name of the language-specific MOF file that will be generated.
-AMENDMENT:<Locale>	Splits the MOF file into language-neutral and -specific versions. The MOF compiler creates a language neutral form of the MOF file that has all amended qualifiers removed. A localized version of the MOF file is also created with an MFL file extension. The Locale parameter specifies the name of the child namespace that contains the localized class definitions. The format of the Locale parameter is MS_*xxx* where *xxx* is the hexadecimal value of the Win32 LCID. For example, the locale for American English is MS_409. For more information, see WMI Localization.
<MOFfile>	Specifies MOFfile as the name of the file to parse.

2.8.5.2 WMI CIM Studio

WMI CIM Studio is an application included in the WMI Tools package. This package is available as a separate download for Windows 2000, Windows XP, and Windows.NET Server from http://www.microsoft.com/

downloads/release.asp?ReleaseID=40804. It uses a Web interface to display information, but it relies on the collection of ActiveX components installed on the system when it runs for the first time. **WMI CIM Studio** provides the ability to do the following:

- Connect to a chosen system and browse the CIM repository in any namespace available
- Search for classes by their names, descriptions, or property names
- Review the properties, methods, and associations related to a given class
- See the instances available for a given class of the examined system
- Perform queries in WQL language
- Generate a MOF file based on selected classes
- Compile a MOF file to load it in the CIM repository

Figures 2.9 and 2.16 show two screen-shot samples of the **WMI CIM Studio**. Each time a class is selected in **WMI CIM Studio**, the right pane shows its properties, methods, and associations. Take a few minutes to play with the **WMI CIM Studio**, as it will help you understand how the CIM repository is organized. Figure 2.21 shows the meaning of the various symbols used by the **WMI CIM Studio**.

In the previous section, we saw how to register the SMTP event consumer that comes with the WMI installation. You can find the *SMTPEventConsumer* class with the **WMI CIM Studio** by performing a search on the namespace used for the registration. You will see that the definition made in the MOF file matches the view of the **WMI CIM Studio** as shown in Figure 2.22.

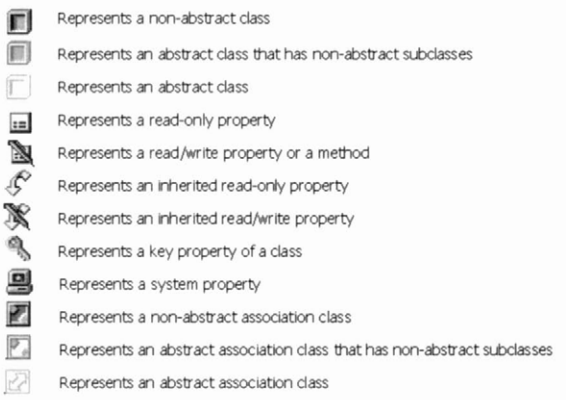

Figure 2.21
The WMI CIM Studio symbols.

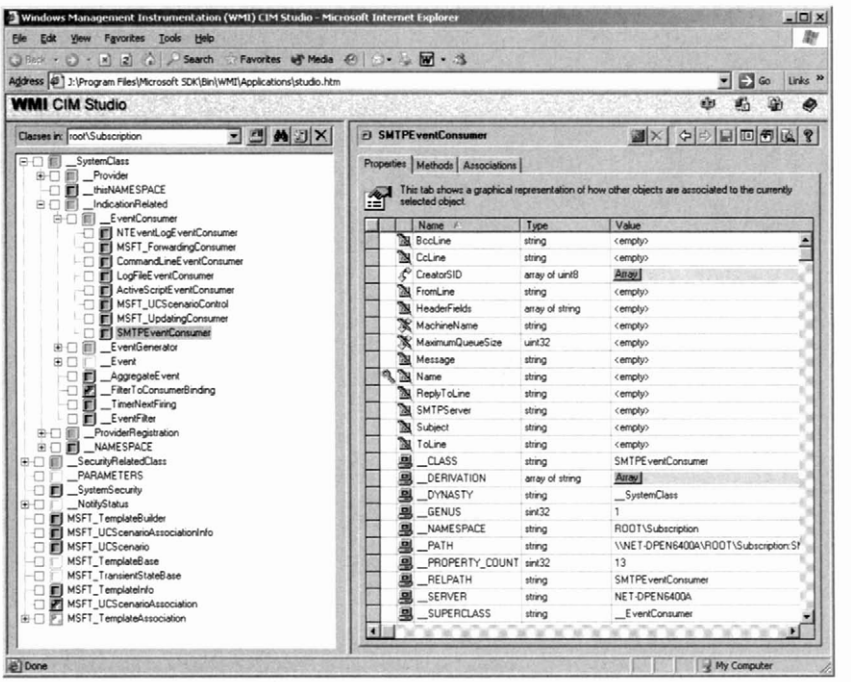

Figure 2.22 *The SMTPEventConsumer class.*

2.8.5.3 *WMI Object Browser*

The **WMI Object Browser** uses the same interface as **WMI CIM Studio**, but instead of browsing classes with related properties and methods, the **WMI Object Browser** browses instances only. Like **WMI CIM Studio**, the **WMI Object Browser** is also included in the WMI Tools package available for download from http://www.microsoft.com/downloads/release.asp?ReleaseID=40804. By default, the **WMI Object Browser** starts from the computer object (see Figure 2.23) and retrieves all instances associated with the computer object. It retrieves associated instances using associations (and references) defined in the CIM repository. The *Win32_ComputerSystem* class has 19 associated classes. When using the **WMI Object Browser** to view the instances of the associated classes, it generates more than 10,000 instances associated with the *Win32_ComputerSystem* instance, which represents a lot of information available from a Windows system.

In addition to all instances displayed, you will find the *Win32_Service* class instances that represent every Windows Service available on the system along with their respective statuses. We use this class in a further exercise in Section 2.9.1 of this chapter.

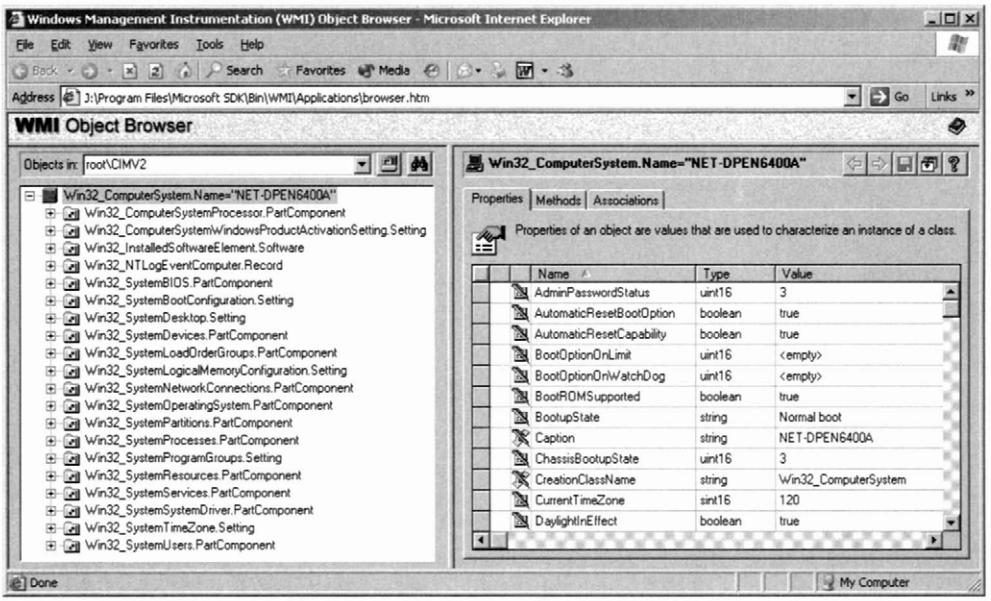

Figure 2.23 *The WMI Object Browser.*

2.8.5.4 WMI event registration

This tool is used to examine the permanent consumer registrations with their associated event filter and consumer instances in the CIM repository. It uses a Web interface to display the information such as **WMI CIM Studio** or the **WMI Object Browser**. It is also included in the WMI Tools package available for download from http://www.microsoft.com/downloads/release.asp?ReleaseID=40804. The tool is designed to add, view, or delete the event registration properties. It supports the consumer, filter, and timer events. This tool is also available from the Platform SDK. We use this tool in a practical exercise in Section 2.9.2 of this chapter.

2.8.5.5 WMI Event Viewer

The WMI Event Viewer is an application to be registered in the CIM repository as a permanent event consumer. Also included in the WMI Tools package available for download from http://www.microsoft.com/downloads/release.asp?ReleaseID=40804, it is an executable file called **WbemEventViewer.exe**. This application can be started from the command prompt, but as a permanent consumer, WMI launches this program if an event is registered for it. This means that the WMI event registration tool can be used to perform this event registration. Note that a MOF file can

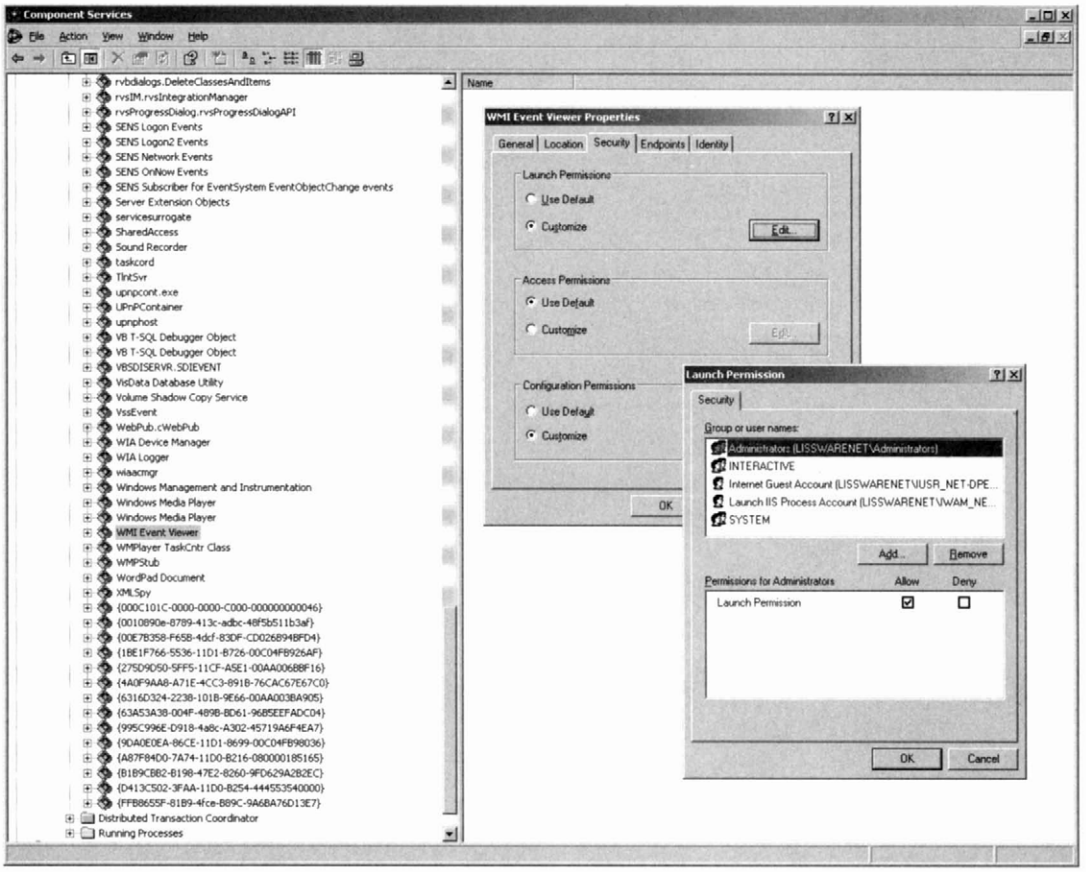

Figure 2.24 *The WMI Event Viewer DCOM security settings.*

also be used to perform this registration. Since a permanent consumer can be launched remotely, this application is registered in a Windows system as a COM/DCOM application and requires specific privileges to be started remotely. You can examine the default security with the **DCOMCNFG.exe** under Windows NT 4.0 and Windows 2000 or with the Component Services MMC snap-in under Windows XP and Windows.NET Server as visible in Figure 2.24.

As a permanent event consumer, the WMI Event Viewer must registered in the CIM repository with a MOF called **Eviewer.mof** shown below:

```
1:// Copyright (c) 1997-1999 Microsoft Corporation
2:#pragma namespace("\\root\\cimv2")
3:
4:
5:// register me as a Physical Event Consumer provider.
```

```
 6:instance of __Win32provider as $P
 7:{
 8:Name = "EventViewerConsumer";
 9:Clsid = "{DD2DB150-8D3A-11d1-ADBF-00AA00B8E05A}";
10:};
11:
12:instance of __EventConsumerproviderRegistration
13:{
14:   provider = $P;
15:   ConsumerclassNames = {"EventViewerConsumer"};
16:};
17:
18:// this is my logical consumer class.
19:[Locale(0x049), UUID("{8502C596-5FBB-11D2-AAC1-006008C78BC7}")]
20:class EventViewerConsumer : __EventConsumer
21:{
22:[key]
23:string Name = "";
24:
25:[read,
26:ValueMap {"0", "1", "2"},
27:      Values {"Error", "Warning", "Information"}]
28:uint8 Severity = 0;          // 0-based.
29:
30:[read]
31:string Description = "";
32:};
```

The **Eviewer.mof** file is included in the WMI Tools package.

A few pages earlier, we saw the MOF file used to register the *SMTP* event consumer. To register the *WMI Event Viewer*, the MOF file uses the same structure. We have the following sections:

- **The Win32 provider section (lines 6 to 10)**: to reference the UUID of the COM/DCOM object implemented by the event consumer provider called **WbemEventViewer.exe**
- **The permanent event consumer provider section (lines 12 to 16)**: to associate the consumer class definition with the consumer provider references
- **The Event Viewer Consumer class definition (lines 20 to 32)**: to define the class representing the consumer provider with its name defined as a key (lines 22 and 23) and two optional parameters: severity with its possible values (lines 25 to 28) and description (lines 30 and 31)

It must be registered with the following command line:

```
C:\>mofcomp -N:ROOT\CIMv2 EViewer.Mof
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: EViewer.Mof
```

```
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

In Section 2.9.2 of this chapter, we will see how to use the WMI Event Viewer with an associated event.

2.8.5.6 WinMgmt.exe

WinMgmt.exe is not a tool; it is the executable that implements the WMI service. Under Windows NT, Windows 2000 and Windows.NET Server, WMI runs as a service. However, under Windows XP and Windows.NET Server, don't expect to find a process called **WinMgmt.exe**, as it runs under the **SvcHost.exe** process. On computers running Windows 98, Windows 95, or Windows Millennium, WMI runs as an application. Under Windows NT, Windows 2000, Windows XP, or Windows.NET Server, it is also possible to run this executable as an application, in which case the executable runs in the current user context. For this, the WMI service must be stopped first. The executable supports some switches that can be useful when starting WMI as a service or as an application. Usually, when scripting with WMI there is no need to run the WMI service as an application. WMI provider developers who may want to debug their providers essentially need to run the WMI service as an application. For completeness, Table 2.6 contains a list of switches supported by **WinMgmt.exe**.

Table 2.6 *WmiMgmt.exe Switches*

/exe	Causes WinMgmt.exe to run as an application rather than as a Windows_NT/Windows_2000 service. You might use this switch only rarely because you will almost always want to run WMI as a service in Windows_NT/Windows_2000. When the /exe switch is used, WinMgmt.exe is run in the users security context. The primary use of the /exe switch in Windows_NT version 4.0 is to make it easier to debug providers. This switch is available on Windows_95, Windows_98, Windows_NT 4.0, and Windows_2000. Using this switch will place an icon in the task bar.
/kill	Terminates all WinMgmt.exe processes on the local system, including WMI processes started as a service by the Service Control Manager or invoked by using the /exe switch. This switch is available on Windows_95, Windows_98, Windows_NT 4.0, and Windows_2000 and later. To use this switch with Windows_NT, you must have administrative rights.
/regserver	Registers the Windows Management Service, adding entries to the operating system registry. The /regserver switch should be implemented by any .exe server. This switch is available on Windows_95, Windows_98, Windows_NT 4.0, and Windows_2000 and later.

Table 2.6 *WmiMgmt.exe Switches (continued)*

/unregserver	Removes registry entries added through the self-registration process and should rarely be used. This switch is available on Windows_95, Windows_98, Windows_NT 4.0, and Windows_2000 and later.
/backup <*filename*>	Causes WMI to back up the repository to the specified file name. The filename argument should contain the full path to the file location. This process requires a write lock on the repository so that write operations to the repository are suspended until the backup process is completed. This switch is available on Windows_95, Windows_98, Windows NT_4.0, and Windows_2000 and later.
/restore <*filename*>	Manually restores the WMI repository from the specified file. The filename argument should contain the full path to the file location. When restoring the repository, the process deletes the existing repository, writes the specified backup file to the automatic backup file, and then connects to WMI to perform the restoration. If exclusive access to the repository cannot be achieved, existing clients are disconnected from WMI. This switch is available on Windows_95, Windows_98, Windows_NT 4.0, and Windows_2000 and later.
/resyncperf <*winmgmt service process id*>	Invokes the AutoDiscovery/AutoPurge (ADAP) mechanism of WMI. For more information on ADAP, see Maintaining Performance Counter Classes. This switch is only available on Windows_2000 and later.
/clearadap	Removes all of the ADAP information from the registry, effectively resetting the state of each performance library. The ADAP utility stores state information about the system performance libraries in the registry. This switch is only available on Windows_2000 and later.

2.8.5.7 WBEMTEST.exe

WBEMTEST.exe is a WMI tester, which is delivered standard with WMI (see Figure 2.25). This tool allows an administrator or a developer to perform most of the tasks from a graphical interface that WMI provides at the API level. Although available under Windows NT, Windows 2000, Windows XP, and Windows.NET Server, this tool is not officially supported by Microsoft. Actually, the use of this tool is not obvious unless you are accustomed to the WMI COM API. It requires a good knowledge of the WMI concepts in relation to the implementation. As we script on top of WMI, we use the features available from the WMI API, which will help clarify **WBEMTEST.exe** usage. Basically, **WBEMTEST.exe** helps to perform the following tasks:

- Enumerate, open, create, and delete classes
- Enumerate, open, create, and delete instances of classes

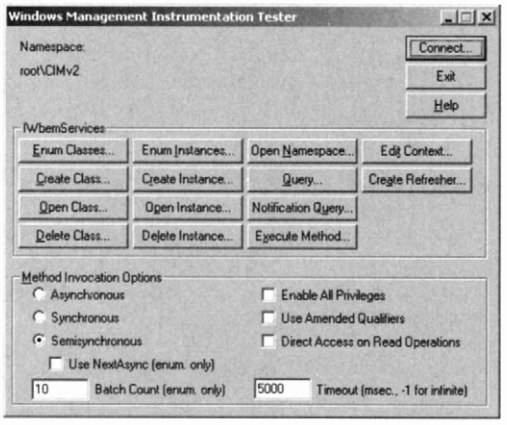

Figure 2.25
The WMI Tester (WBEMTEST.exe)

- Select a namespace
- Perform queries
- Perform queries in relation to events (notification queries)
- Execute methods associated with instances
- Execute every WMI operation asynchronously, synchronously, or semi-asynchronously (we examine this type of operation when we talk about the WMI scripting in Chapters 4, 5, and 6)

For example, if you want to obtain a list of the Windows services with **WBEMTEST.exe**, you can do the following:

1. Connect to the **Root\CIMv2** namespace. If you are not running as an administrator, make sure that you provide the right credentials to access the namespace. By default, only the members of the Administrators group have full rights to the WMI namespaces.
2. Select the "Enum Instances" button and enter the *Win32_Service* class name representing a Windows service and select "immediate only."
3. Once done, you will see an output like that shown in Figure 2.26.

If you use the **WMI CIM Studio**, you can do the same thing. With CIM Studio, you can also see that the *Win32_Service* is a subclass of *Win32_BaseService*, and *Win32_BaseService* is a subclass of the *CIM_Service* class. With **WBEMTEST.exe**, if you perform the same operation as before with the *CIM_Service* class and choose the "Recursive" option, you retrieve all the instances of the class derived from the *CIM_Service* class. In this list, you retrieve the *Win32_Service* instances among other *Win32_SystemDriver*

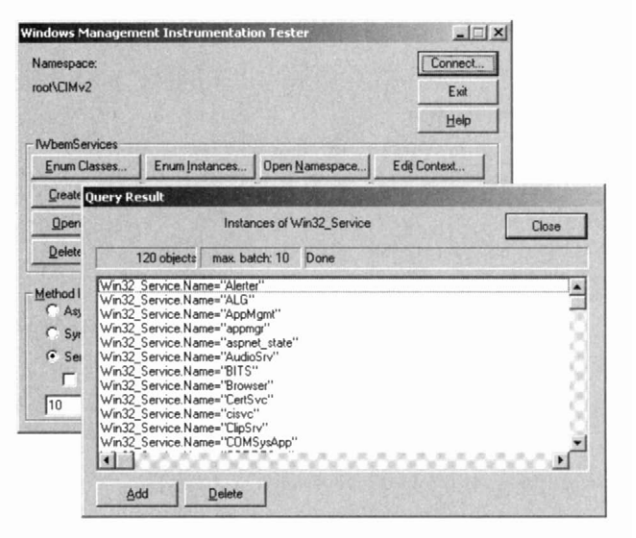

Figure 2.26 *Viewing the Windows Services by enumerating the Win32_Service class instances with WBEMTEST.exe.*

instances, since *Win32_Service* and *Win32_SystemDriver* are subclasses of the *CIM_Service* class.

To give another sample, it is also possible to retrieve the Windows service list by performing a query with **WBEMTEST.exe**. Proceed as follows:

1. Connect to the **Root\CIMv2** namespace. If you are not running as an administrator, make sure that you provide the right credentials to access the namespace.
2. Select the "Query" button and enter the following query string "Select * from Win32_Service" as shown in Figure 2.27.
3. The output of Figure 2.27 will be the same as Figure 2.26.

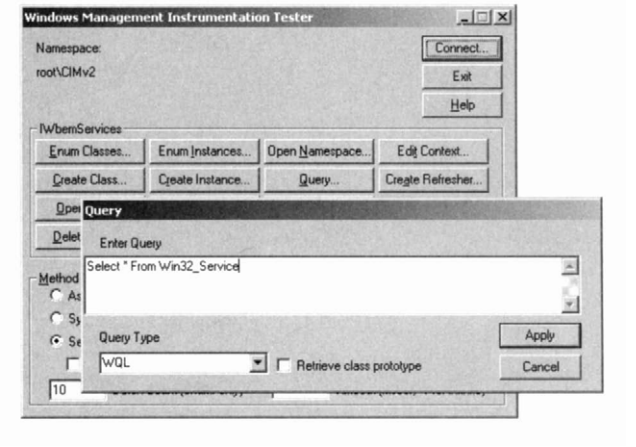

Figure 2.27 *Viewing the Windows Services by querying the Win32_Service class instances with WBEMTEST.exe.*

For a few more examples, try entering the queries shown in Section 2.8.4 used to explain the WMI query language. The data queries and schema queries must be executed via the "Query" button shown in Figure 2.25. The event query must be executed with the "Notification Query" button. This tool is very useful when working with WMI. As your knowledge about WMI increases, the understanding of **WBEMTEST.exe** becomes even clearer.

2.8.5.8 The WMI command-line tool: WMIC

WMIC is a command-line tool designed to ease WMI information retrieval from a system by using some simple keywords (aliases). **WMIC.exe** is available only under Windows XP Professional and Windows.NET Server. It is not included in other Windows platforms. By typing "WMIC /?" at the command line, you can obtain a complete list of the switches and reserved keywords available. Table 2.7 summarizes the global switches. By typing "WMIC switch-name /?", you can gather more information about the switch usage.

Table 2.8 summarizes the reserved keywords (called the WMIC aliases). By typing "WMIC alias /?", you can gather more information about the alias usage.

Table 2.9 shows some specific commands. By typing "WMIC (class | path | context) /?", you can gather more information about the alias usage.

Table 2.7 *WMIC Global Switches*

/NAMESPACE	Path for the namespace the alias operates against
/ROLE	Path for the role containing the alias definitions
/NODE	Servers the alias will operate against
/IMPLEVEL	Client impersonation level
/AUTHLEVEL	Client authentication level
/LOCALE	Language ID the client should use
/PRIVILEGES	Enables or disables all privileges
/TRACE	Outputs debugging information to stderr
/RECORD	Logs all input commands and output
/INTERACTIVE	Sets or resets the interactive mode
/USER	User to be used during the session
/PASSWORD	Password to be used for session login
/?[:<BRIEF|FULL>	Usage information

Table 2.8 *WMIC Aliases*

BASEBOARD	Provides access to the baseboard (also known as a motherboard or system board)
BIOS	Provides access to the attributes of the computer system's basic input/output services (BIOS) that are installed on the computer
BOOTCONFIG	Provides access to the boot configuration of a system
CDROM	Provides access to the CD-ROM drives on a computer system
COMPUTERSYSTEM	Provides access to the computer system operating in a user environment
CSPRODUCT	Corresponds to software and hardware products used on a computer system
DEVICEMEMORYADDRESS	Provides access to the device memory addresses on a system
DIRECTORY	Provides access to the directory entries on a computer system
DISKDRIVE	Describes a physical disk drive as seen by a computer
DMACHANNEL	Provides information on the direct memory access (DMA) channel on a computer system
DRIVERVXD	Provides access to the virtual device driver/s on a computer system
ENVIRONMENT	Provides access to the system environment settings on a computer system.
GROUP	Provides access to data about a group account
IRQRESOURCE	Provides access to the interrupt request line (IRQ) number on a computer system
LOADORDER	Provides access to the group of system services that define execution dependencies
LOGICALDISK	Provides access to the data source that resolves to an actual local storage device on a system
LOGICALMEMORY	Allows access to the configuration layout and examination of the availabe memory on a system
NETADAPTER	Provides access to the network adapters installed on a system
NETADAPTERCONFIG	Provides access to and allows changing the attributes and behaviors of a network adapter
NETCONNECTION	Provides access to the active network connection in an environment
NETLOGIN	Provides access to the network login information of a particular user on a system
NETPROTOCOL	Provides access to the protocols and their network characteristics on a computer system
NTEVENTLOG	Allows access to the NT eventlog file
ONBOARDDEVICE	Describes common adapter devices built into the motherboard (system board)
OS	Provides access to the operating system/s installed on a computer
PAGEFILE	Provides access to the files used for handling virtual memory file swapping on a system
PAGEFILESET	Provides access to the settings of a page file
PARTITION	Provides access to the capabilities and management capacity of a partitioned area of a physical disk on a system

Table 2.8 *WMIC Aliases (continued)*

PHYSICALMEMORY	Allows access to details about the computer system's physical memory
PORT	Provides access to the I/O ports on a computer system
PORTCONNECTOR	Provides access to the physical connection ports
PRINTER	Provides access to all printer devices connected to a computer system
PRINTERCONFIG	Allows review of the configuration for a printer device
PRINTJOB	Provides access to the print jobs generated by an application
PROCESS	Provides access to the sequence of events on a system
PRODUCT	Correlates tasks to a single installation package
RECOVEROS	Provides access to the types of information that will be gathered from memory when the operating system fails
REGISTRY	Provides access to the computer system registry
SCHEDULEDJOB	Provides access to the jobs scheduled using the schedule service
SERVER	Provides access to the server information
SERVICE	Provides access to the service applications on a computer system
SHARE	Allows access to the shared resources on a system
SOFTWAREELEMENT	Provides access to the elements of a software product installed on a system
SOFTWAREFEATURE	Provides access to and allows changing of the software product subsets of SoftwareElements
STARTUP	Allows access to the command that runs automatically when a user logs onto the computer system
SYSACCOUNT	Allows access to the system accounts
SYSDRIVER	Provides access to the system driver for a base service
SYSTEMENCLOSURE	Provides access to the properties associated with a physical system enclosure
SYSTEMSLOT	Provides access to the physical connection points including ports, slots and peripherals, and proprietary connections points
TAPEDRIVE	Describes a tape drive on a computer
TEMPERATURE	Allows access to the properties of a temperature sensor (electronic thermometer)
TIMEZONE	Provides access to the time-zone information for a system
UPS	Provides access to the capabilities and management capacity of an uninterruptible power supply (UPS)
USERACCOUNT	Provides access to information about a user account on a system
VOLTAGE	Allows access to the properties of a voltage sensor (electronic voltmeter)
WMISET	Provides access to and allows changes to be made to the operational parameters for the WMI service

Table 2.9 *WMIC Commands*

CLASS	Escapes to full WMI schema
PATH	Escapes to full WMI object paths
CONTEXT	Displays the state of all global switches
QUIT/EXIT	Exits the program

Basically, the WMIC interoperates with existing shells and utility commands. When starting WMIC for the first time, Windows installs it on the system and displays a command prompt:

```
wmic:root\cli>
```

From this prompt all aliases of Table 2.8 can be entered interactively. In the previous section, to retrieve the list of the Windows Services we used **WBEMTEST.exe**. This required use of the "Enum instances" button and the reference to the *Win32_Service* class (or a WQL data query like "Select * From Win32_Service" and the use of the "Query" button). With WMIC, it is possible to obtain the same type of information by using an alias (see Table 2.8), which greatly facilitates the command syntax. To locate, all *Win32_Service* instances with WMIC, you must use the *service* alias from the WMIC command prompt (interactive mode) or from the command line (noninteractive mode):

```
C:\>wmic service
AcceptPause AcceptStop Caption                                    CheckPoint CreationclassName →
FALSE       TRUE       Alerter                                    0          Win32_Service     →
FALSE       FALSE      Application Management                     0          Win32_Service     →
FALSE       TRUE       Windows Audio                              0          Win32_Service     →
FALSE       FALSE      Background Intelligent Transfer Service    0          Win32_Service     →
FALSE       TRUE       Computer Browser                           0          Win32_Service     →
TRUE        TRUE       Certificate Services                       0          Win32_Service     →
FALSE       FALSE      Indexing Service                           0          Win32_Service     →
FALSE       FALSE      ClipBook                                   0          Win32_Service     →
FALSE       FALSE      COM+ System Application                    0          Win32_Service     →
FALSE       TRUE       Cryptographic Services                     0          Win32_Service     →
FALSE       TRUE       Distributed File System                    0          Win32_Service     →
FALSE       TRUE       DHCP Client                                0          Win32_Service     →
FALSE       FALSE      DHCP Server                                0          Win32_Service     →
FALSE       FALSE      Logical Disk Manager Administrative Service 0         Win32_Service     →
FALSE       TRUE       Logical Disk Manager                       0          Win32_Service     →
TRUE        TRUE       DNS Server                                 0          Win32_Service     →
FALSE       TRUE       DNS Client                                 0          Win32_Service     →
FALSE       FALSE      Event Log                                  0          Win32_Service     →
FALSE       TRUE       COM+ Event System                          0          Win32_Service     →
FALSE       FALSE      Fast User Switching Compatibility Services 0          Win32_Service     →
FALSE       TRUE       Help and Support                           0          Win32_Service     →
...         ...        ...                                        ...        ...               →
...         ...        ...                                        ...        ...               →
...         ...        ...                                        ...        ...               →
```

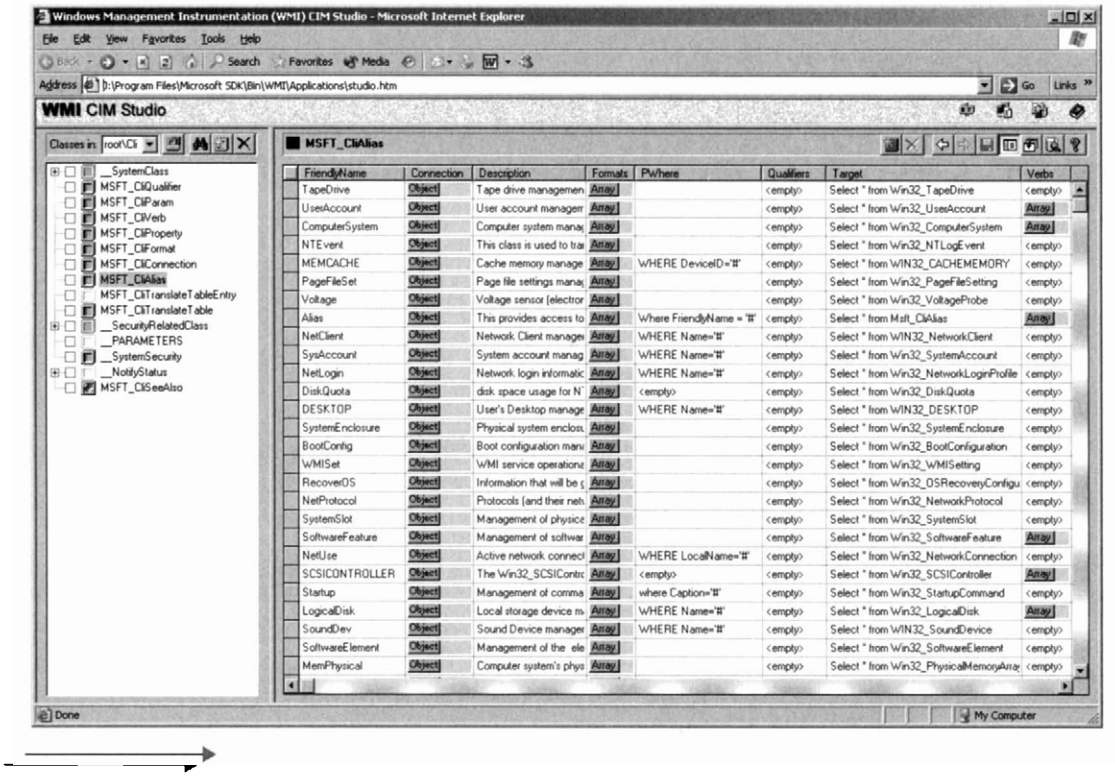

Figure 2.28 *WMIC alias instances.*

The above example all instances of the *Win32_Service* class are retrieved with the *service* alias. The aliases in Table 2.8 are nothing more than an abstraction of the syntax and the class name used to retrieve this information. Aliases are used to capture the features of a WMI class relevant to a specific task, such as disk or network administration. As shown in Figure 2.28, it is interesting to know that all WMIC aliases are stored in the CIM repository as instances of the *MSFT_CliAlias* class in the **Root\Cli namespace**. Therefore, by creating new instances of the *MSFT_CliAlias* class, it possible to extend the set of aliases available to WMIC.

In the previous example, WMIC accessed all instances of the *Win32_Service* class locally, but WMIC can access any WMI information on any WMI-enabled system. Therefore, WMIC is not required on the remote system, which makes it usable in a heterogeneous environment (e.g., Windows 2000, Windows XP, and Windows NT computers). With the **/Node** switch you can access a remote computer (see Table 2.7).

```
C:\>wmic /Node:"NET-DPEP6400.Emea.LissWare.NET" service
AcceptPause AcceptStop Caption                                  CheckPoint CreationclassName →
FALSE       TRUE       Alerter                                  0          Win32_Service     →
FALSE       FALSE      Application Management                   0          Win32_Service     →
FALSE       TRUE       Windows Audio                            0          Win32_Service     →
FALSE       FALSE      Background Intelligent Transfer Service  0          Win32_Service     →
```

As another example, the **/Trace** switch enables the error tracing to returns error information for every executed command. The complete list of global switches is available in Table 2.7.

As we have seen, WMIC makes use of aliases, but it also uses verbs. Every alias has a series of supported verbs. For example, by typing

```
C:\>wmic service /?

SERVICE - Service application management.

HINT: BNF for Alias usage.
(<alias> [WMIObject] | <alias> [<path where>] | [<alias>] <path where>) [<verb clause>].

USAGE:

SERVICE ASSOC [<format specifier>]
SERVICE CALL <method name> [<actual param list>]
SERVICE CREATE <assign list>
SERVICE DELETE
SERVICE GET [<property list>] [<get switches>]
SERVICE LIST [<list format>] [<list switches>]
```

you can see the list of verbs related to the *service* alias. A verb represents an action that you can take for the specified alias. Basically, all verbs are the same for all aliases, but some of them are not supported by some aliases. For instance, if you compare the verbs of the *service* alias with the verbs available from the *bios* alias, you will see that the *call* verb is not available for the *bios* alias. For example, to gather information about the disk configuration of a remote system, we execute WMIC with the *logicaldisk* alias and the *get* verb to retrieve a limited set of properties from the class representing the logical disks (i.e., *Win32_LogicalDisk*):

```
C:\>wmic /Node:"net-dpen6400.lissware.net"
        logicaldisk get description,DeviceID,FileSystem,FreeSpace,Size,VolumeDirty,VolumeName
Description             DeviceID  FileSystem  FreeSpace   Size        VolumeDirty  VolumeName
3 1/2 Inch Floppy Drive  A:
Local Fixed Disk         C:        NTFS        821909504   843816448   FALSE        Boot
Local Fixed Disk         D:        NTFS        1094373376  3220406272  FALSE        APPS
Local Fixed Disk         E:        FAT         514031616   533848064   FALSE        DEV
Local Fixed Disk         F:        NTFS        763207680   1586962432  FALSE        SOURCES
Local Fixed Disk         G:        NTFS        172371968   3777789952  FALSE        DATA
Local Fixed Disk         H:        FAT         285442048   1416429568  FALSE        GAMES
Local Fixed Disk         I:        NTFS        2541527040  9932234752  FALSE        Windows 2000
Local Fixed Disk         J:        NTFS        1150111744  9932234752  FALSE        Windows.NET
CD-ROM Disc              L:
```

Next, as a logical disk can't exist without a disk partition, there is an association created in the CIM repository between logical disk instances and disk partitions. Therefore, with WMIC it is possible to view these associations. In this example, we examine the associations in place only by using the *assoc* verb and the where statement to specifically examine the C: logical disk:

```
C:\>wmic /node:"net-dpen6400.LissWare.NET" logicaldisk where deviceid="c:" assoc
__PATH
\\NET-DPEN6400A\ROOT\CIMV2:Win32_LogicalDisk.DeviceID="C:"                                    →
\\NET-DPEN6400A\ROOT\CIMV2:Win32_Directory.Name="c:\\"                                         →
\\NET-DPEN6400A\ROOT\CIMV2:Win32_ComputerSystem.Name="NET-DPEN6400"                            →
\\NET-DPEN6400A\ROOT\CIMV2:Win32_QuotaSetting.VolumePath="C:\\"                                →
\\NET-DPEN6400A\ROOT\CIMV2:Win32_DiskPartition.DeviceID="Disk #0, Partition #0"                →
\\NET-DPEN6400A\ROOT\CIMV2:Win32_SystemAccount.Domain="NET-DPEN6400",Name="Administrators"     →
```

As we can see, the C: logical disk is associated with the Disk#0, Partition #0" disk partition. Therefore, with WMIC we can gather information about the partition itself with the *partition* alias and the where statement to specifically examine the "Disk#0, Partition #0 disk partition:

```
C:\>wmic /node:"net-dpen6400.LissWare.NET" partition where
                DeviceID="Disk #0, Partition #0" get "bootpartition,description,deviceid,bootable"
Bootable  BootPartition  Description     DeviceID
TRUE      TRUE           MS-DOS V4 Huge  Disk #0, Partition #0
```

With this WMIC command, we see that the C: logical drive located on the "Disk #0, Partition #0" is a bootable logical disk.

If you need to restart a service on a remote system, you can use the service alias combined with the *call* verb. For example, to restart the SNMP service, the WMIC command line to use is as follows:

```
C:\>wmic /node:"net-dpen6400.LissWare.NET" service where name='snmp' call stopservice
Executing (\\NET-DPEN6400\ROOT\CIMV2:Win32_Service.Name="SNMP")->stopservice()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ReturnValue = 0;
};

C:\>wmic /node:"net-dpen6400.LissWare.NET" service where name='snmp' call startservice
Executing (\\NET-DPEN6400\ROOT\CIMV2:Win32_Service.Name="SNMP")->startservice()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ReturnValue = 0;
};
```

An interesting feature of WMIC is the ability to request an output result in HTML. The only thing you need is a stylesheet (XSL file) to generate the HTML content. WMIC comes with more than 10 XSL files available from

%SystemRoot%\System32\WBEM. Each XSL files generates a specific output format. For instance, if we execute WMIC with the *service* alias and specify the **/Translate** switch to request the WMI information in XML and specify the **/Format** switch to define which XSL file to use for the transformation, we will generate the service list obtained in the beginning of this section in HTML:

```
C:\>wmic /Output:ServiceList.HTM service list /translate:basicxml /Format:htable.xsl
```

Htable.xsl is one of the stylesheets that comes with WMIC. The obtained result is shown in Figure 2.29.

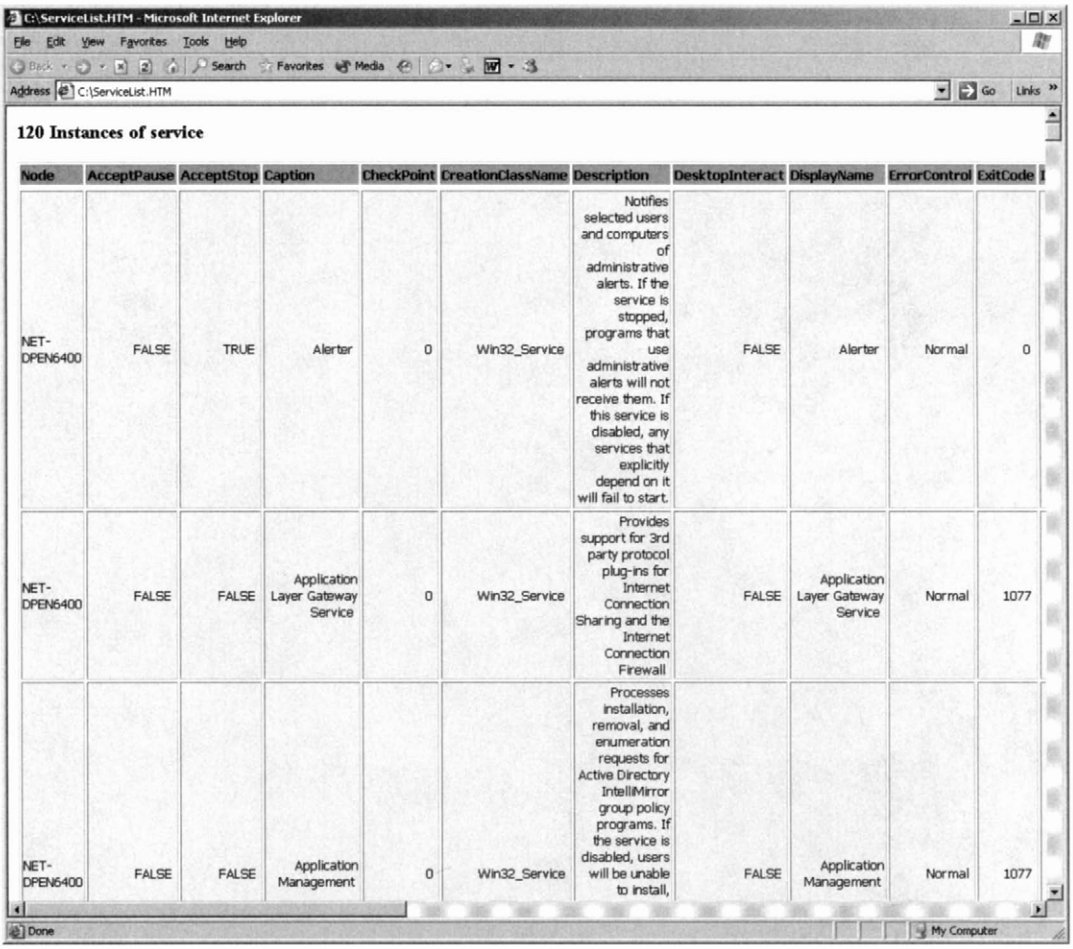

Figure 2.29 *An HTML file created WMIC to show the service list.*

To give another example, instead of generating a list of services, we can generate a simple form containing information about one service. Therefore, the WMIC command line will look like:

```
C:\>wmic /Output:ServiceSNMP.HTM service where
name='snmp' get /translate:basicxml /Format:hform.xsl
```

Hform.xsl is another stylesheet that comes with WMIC. The obtained result is shown in Figure 2.30.

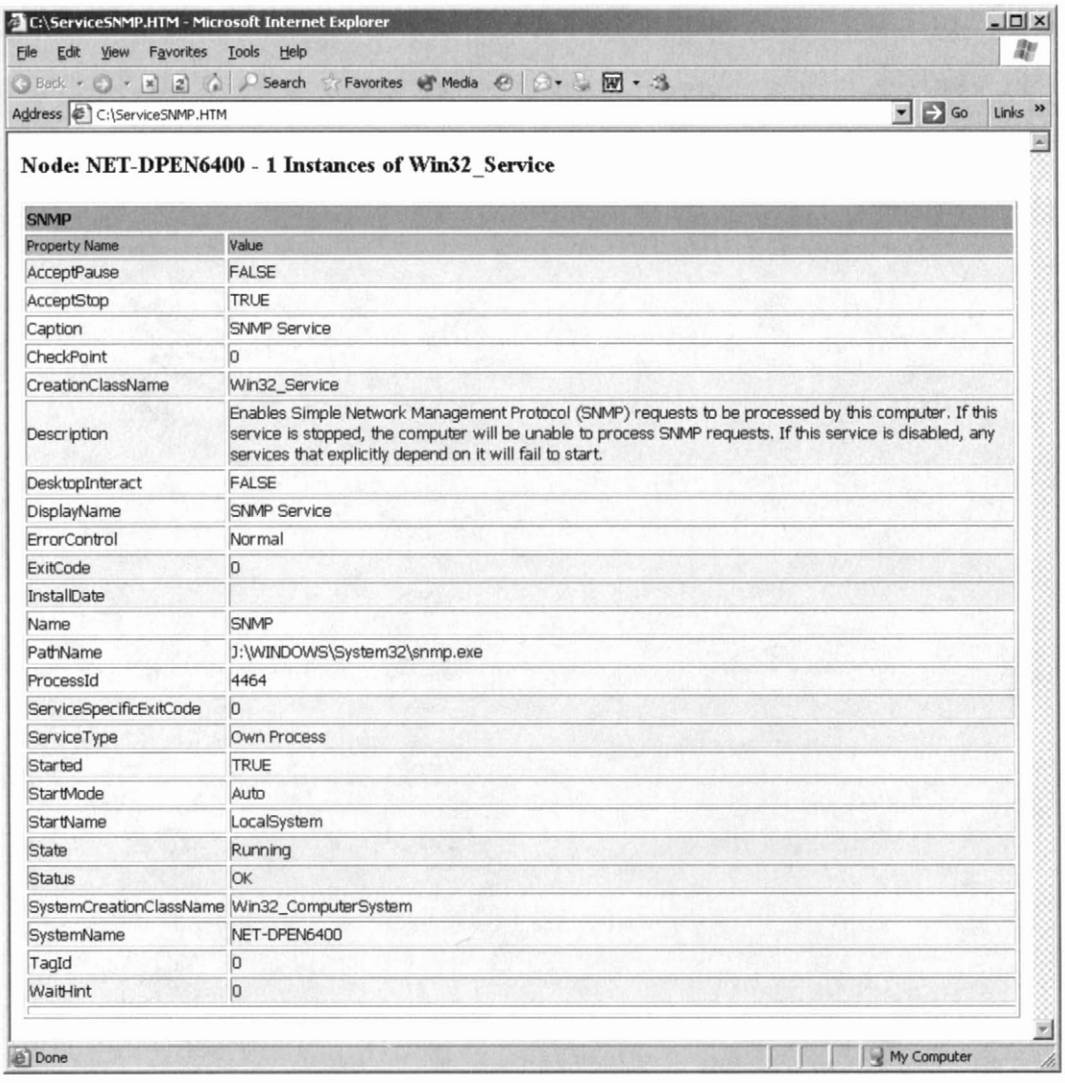

Figure 2.30 *An HTML file created WMIC to show service properties.*

Table 2.10 *Available WMIC Style Sheets*

WMIC Transformations	
csv.xsl	Generate a Comma Separated Value (CSV) file.
hform.xsl	Generate an HTML table with an item per form (See Figure 2.30).
htable.xsl	Generate an HTML table with an item per row (See Figure 2.29).
mof.xsl	Generate a MOF file.
rawxml.xsl	Generate a raw XML file.
texttable.xsl	Generate an ASCII table
textvaluelist.xsl	Generate an ASCII file containing the properties with their values.
xml.xsl	Generate an XML file.

Table 2.10 summarizes the various style sheets available.

We have now discovered some of the features available from WMIC. WMIC can be a good alternative to a WSH script development and is therefore a valuable tool to consider in some situations. It greatly facilitates access to the information provided by WMI without requiring specific WMI knowledge. It is an important addition to Windows XP and Windows.NET Server for the administrator not practicing WMI every day and for WMI beginners.

2.8.5.9 WBEMDUMP.exe

WBEMDUMP is a tool delivered with the Platform SDK. This command-line tool comes with its own Visual C++ project. Basically, the tool can show the CIM repository classes, instances, or both. It is possible to retrieve the same information as that retrieved with WMIC. However, **WBEMDUMP.Exe** requires more specific knowledge about WMI, as it doesn't abstract WMI as WMIC. However, it runs under Windows NT 4.0 and Windows 2000. It is also possible to execute methods exposed by classes or instances. Even if it is not a standard WMI tool delivered with the system installation, this tool can be quite useful for exploring the CIM repository. For example, to browse from the **Root** namespace across all namespaces and list all classes available without their properties, use the following command line:

```
C:\>WBEMDUMP /S /S2 /D Root
<ROOT>
  <ROOT\CIMV2>
    MSFT_WMI_NonCOMEventprovider
```

```
    EventViewerConsumer
    TriggerEventConsumer
    SMTPEventConsumer
            Win32_PowerManagementEvent
            Win32_DeviceChangeEvent
                    Win32_SystemConfigurationChangeEvent
                    Win32_VolumeChangeEvent
            MSFT_WMI_GenericNonCOMEvent
            MSFT_NCProvEvent
                    MSFT_NCProvNewQuery
                    MSFT_NCProvAccessCheck
                    MSFT_NCProvCancelQuery
                    MSFT_NCProvClientConnected
            Win32_ComputerSystemEvent
                    Win32_ComputerShutdownEvent
            MSFT_ForwardedMessageEvent
                    MSFT_ForwardedEvent
            MSFT_WmiSelfEvent
                    Msft_Wmiprovider_OperationEvent
                            Msft_Wmiprovider_ComServerLoadOperationEvent
                            Msft_Wmiprovider_InitializationOperationFailureEvent
                            Msft_Wmiprovider_LoadOperationEvent
                            Msft_Wmiprovider_OperationEvent_Pre
                                    Msft_Wmiprovider_DeleteclassAsyncEvent_Pre
...
```

To get a complete list of the options available, run **WBEMDUMP.exe** as follows:

```
C:\>wbemdump /?
WBEMDUMP - Dumps the contents of the CIMOM database.

Syntax: wbemdump [switches] [Namespace [class|ObjectPath] ]
        wbemdump /Q [switches] Namespace QueryLanguage Query

Where:  'Namespace' is the namespace to dump (defaults to root\default)
        'class' is the name of a specific class to dump (defaults to none)
        'ObjectPath' is one instance (ex "SclassA.KeyProp=\"foobar\"")
        'QueryLanguage' is any WBEM supported query language (currently only
           "WQL" is supported).
        'Query' is a valid query for the specified language, enclosed in quotes
        'switches' is one of
           /S Recurse down the tree
           /S2 Recurse down Namespaces (implies /S)
           /E Show system classes and properties
           /E1 Like /E except don't show __SERVER or __PATH property
           /E2 Shows command lines for dumping instances (test mode)
           /D Don't show properties
           /G Do a GetObject on all enumerated instances
           /G2 Do a GetObject on all reference properties
           /G:<x> Like /G using x for flags (Amended=131072)
           /M Get class MOFS instead of data values
           /M2 Get Instance MOFS instead of data values
           /M3 Produce instance template
           /B:<num> CreateEnum flags (SemiSync=16; Forward=32)
           /W  Prompt to continue on warning errors
           /WY Print warnings and continue
           /W:1 Use IWbemclassObject::GetObjectText to show errors
           /H:<name>:<value> Specify context object value (test mode)
           /CQV:<name>[:value] Specify a class qualifier on which to filter
           /T Print times on enumerations
           /T2 Print times on enumerations using alternate timer
           /O:<file> File name for output (creates Unicode file)
```

```
            /C:<file> Command file containing multiple WBEMDUMP command lines
            /U:<UserID> UserID to connect with (default: NULL)
            /P:<Password> Password to connect with (default: NULL)
            /A:<Authority> Authority to connect with
            /I:<ImpLevel> Anonymous=1 Identify=2 Impersonate=3(dflt) Delegate=4
            /AL:<AuthenticationLevel> None=1 Conn=2 Call=3 Pkt=4 PktI=5 PktP=6
            /Locale:<localid> Locale to pass to ConnectServer
            /L:<LoopCnt> Number of times to enumerate instances (leak check)

Notes:  - You can redirect the output to a file using standard redirection.
        - If the /C switch is used, the namespace on the command line must
          be the same namespace that is used for each of the command lines.
          It is not possible to use different namespaces on the different lines
          in the command file.

EXAMPLES:

  WBEMDUMP /S /E root\default            - Dumps everything in root\default
  WBEMDUMP /S /E /M /M2 root\default     - Dump all class & instance mofs
  WBEMDUMP root\default foo              - Dumps all instances of the foo class
  WBEMDUMP root\default foo.name=\"bar\" - Dumps one instance of the foo class
  WBEMDUMP /S2 /M root    - Dumps mofs for all non-system classes in all NS's
  WBEMDUMP /Q root\default WQL "SELECT * FROM Environment WHERE Name=\"Path\""
```

You can refer to the platform SDK to get more information about this tool. We also use this tool when working with data queries and schema queries in Chapter 3. This will give us a chance to practice using the tool and its feature.

2.8.5.10 Scriptomatic

Scriptomatic is a very handy tool released by Microsoft in August 2002. It can be downloaded from http://www.microsoft.com/downloads/release.asp?ReleaseID=41528. This tool is a hypertext application (HTA extension) loading all Win32 WMI classes available from the **Root\CIMv2** namespace. Based on the class selected on the user interface, it generates a VBScript able to list all instances of the class with their properties and values (see Figure 2.31).

In Chapter 4, we will examine the WMI scripting interfaces available and explain in detail the various aspects of WMI scripts similar to the one generated by Scriptomatic.

The main purpose of Scriptomatic is to generate a basic script. The script's intended purpose, in this first release, is to provide information about instances of Win32 classes. Nothing more! Therefore, Scriptomatic will not generate a fully customized script handling complete management business logic. However, for a beginner at WMI scripting, it can help a user to discover properties of the Win32 WMI classes and how to use the WMI API to enumerate properties and values. Therefore, although handy, Scriptomatic does not enable you to learn how to script on top of WMI and how to make use of the WMI class capabilities. Once you get used to the WMI

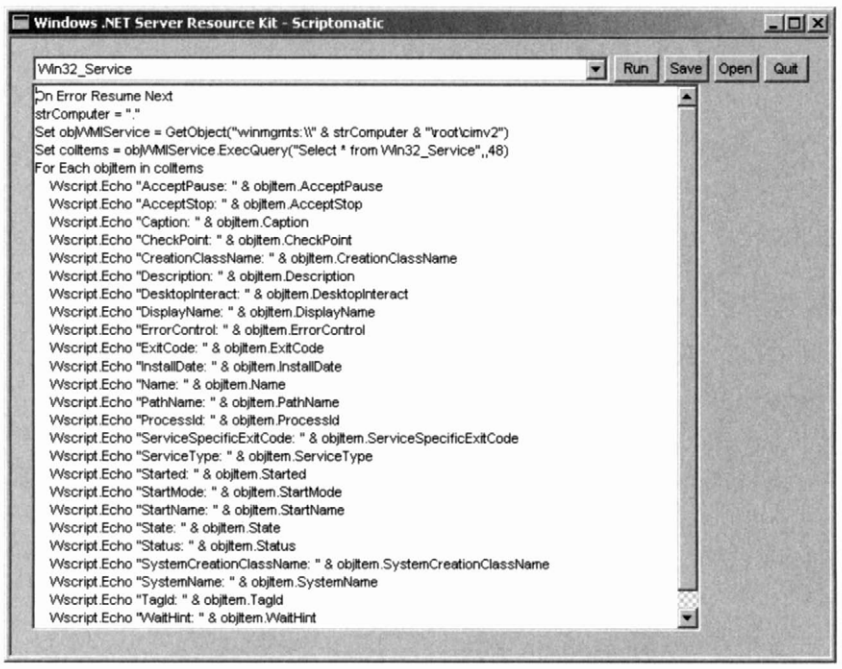

Figure 2.31
The Scriptomatic window.

scripting technique, you will see that the challenge does not reside in the scripting technique in itself but in knowledge of the WMI classes available and the features they provide.

When using Scriptomatic, keep in mind that the following limitations apply:

- **Scriptomatic is designed to work with only a subset of classes, the Win32 classes that return property values:** Only the classes that return instances, located in the **Root\CIMv2** namespace and having a name starting with Win32_ (i.e., Win32_Service) are supported by Scriptomatic.
- **Scriptomatic won't return values for all properties: Some classes return properties in the form of an array.** Scriptomatic does not return array properties in this first release. We will see throughout Chapter 4 how to examine array properties.
- **Scriptomatic won't interpret returned values for you:** Some properties return a value to reflect an instance status. We will see throughout Chapter 4 how to interpret values returned by properties.
- **Scriptomatic can only return property values; it can't be used to run methods:** Scriptomatic is a tool generating passive scripts returning

information about classes. However, Scriptomatic does not generate scripts performing actions on instances of classes. Once more, we will see throughout Chapters 4 and 5 how to handle this from scripts.

- **Scriptomatic works only with WMI:** We will see in Chapter 5 that WMI also integrates with other technologies such as Active Directory Service Interfaces (ADSI). Scriptomatic focuses on WMI script creation only.

By learning the WMI scripting technique presented in the next chapters, we will see how to overcome the current limitations of Scriptomatic and how to understand WMI scripting techniques (Volume 1, *Understanding WMI Scripting*, ISBN 1555582664) and leverage WMI scripting (Volume 2, *Leveraging WMI Scripting*, ISBN 15555582990) to exploit the full power of WMI under Windows. However, as a start, it is a handy tool to play with. Don't wait to be an expert in WMI scripting; it can help you right now to work with some basic scripts.

2.9 Let's be more practical

Now that we have discovered the list of the WMI tools provided by Microsoft and the new vocabulary associated with the CIM repository and its WMI implementation, we are ready to take a closer look at this instrumentation and how it maps to the real world. As we said before, the CIM repository represents objects from the real world. Of course, after reviewing the terms commonly used in the WBEM world, this seems to be sometimes far from the real-world manageable entities. Some terms may still look a bit abstract and seem to be quite hard to apply to a real, manageable world. The purpose of the next two sections is to give a more practical view of these terms in relation to things that every Windows administrator knows. Most of the WMI or WBEM descriptions available today start with a highly descriptive theory, which is quite hard to assimilate. Although some theory is necessary, we examine the components that every Windows administrator is used to working with in a Windows system and see how WMI represents them.

In the first section we use the Windows service as sample. This exercise will directly map most of the WBEM and WMI terms discovered in a real-world manageable entity: a Windows service with its dependencies. In the second section, we create a usable application without programming by simply using some WMI features available out of the box. This exercise will explain the role of a WMI event consumer, the role of a MOF file, and the usefulness of the WMI query language (WQL). As you will see, when we use WQL statements, some small details will probably be not very clear in

some places. Don't worry; we will revisit this topic in Chapter 3. For now, we are in the early stages of our WMI discovery. To get the full benefit of these two exercises, it is strongly recommended that you have a Windows 2000/Exchange 2000 Server system up and running to practice.

2.9.1 Understanding dependencies between classes

A very good example of dependencies is represented by the Windows services dependencies. To avoid some situations in which one Windows service can start before another, the Service Manager makes use of the following registry keys with their related meanings:

- **Group (REG_SZ) group name:** Specifies the name of the group of which the particular service is a member.
- **DependOnGroup (REG_MULTI_SZ) group name:** Specifies zero or more group names. If one or more groups are listed, at least one service from the named group must be loaded before this service is loaded.
- **DependOnService (REG_MULTI_SZ) service name:** Specifies zero or more services names. If a service is listed here, that named service must be loaded before this service is loaded.

A good example of a service having dependencies is the Microsoft Exchange System Attendant (named *MSExchangeSA* in the registry) shown in Figure 2.32.

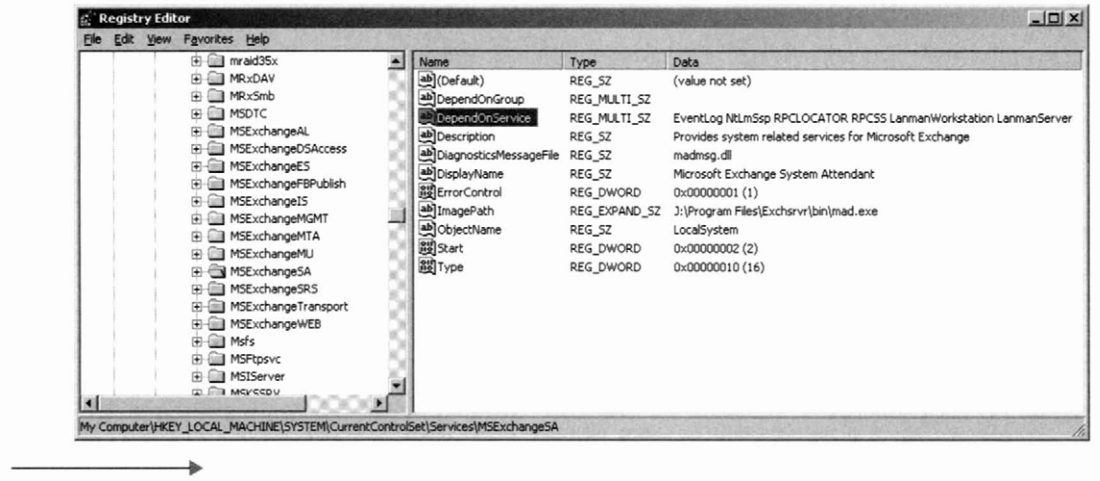

Figure 2.32 *The Microsoft Exchange 2000 System Attendant registry keys.*

This service depends on several other services, named *EventLog*, *NtLmSsp*, *RPCLOCATOR*, *RPCSS*, *LanmanWorkstation*, and *LanmanServer*. When using **WMI CIM Studio**, you can view the same information when starting from the *Win32_Service* class. Proceed as follows:

1. Connect to the **Root\CIMv2** namespace.
2. Expand the classes *CIM_ManagedSystemElement*, *CIM_LogicalElement*, *CIM_Service*, and *Win32_BaseService*.
3. Select the *Win32_Service* class.
4. Request to view instances of this service class by pressing the "view instances" button in the button bar located in the top right pane of **WMI CIM Studio**.
5. **WMI CIM Studio** shows the list of Windows services available in your system. Browse the list until you find the *MSExchangeSA* service.
6. Right click on it and select "Go to object." Now you have the properties of the *MSExchangeSA* service only.
7. Select the "Associations" tab and you should get the Figure 2.33.

Besides the computer system where this service is running, we clearly see that **WMI CIM Studio** shows a list of Windows service instances. A portion of the services listed corresponds exactly to the list set in the registry in the *DependOnService* key. In the list of services, you will notice that **WMI CIM Studio** lists the *MSExchangeMTA* and *MSExchangeIS* even though they are not listed in the *DependOnService* registry key. These two services are also associated with the *MSExchangeSA*, but the relation is not the same as with the other services. *MSExchangeSA* relies, for example, on the *LanManWorkstation* and the *LanManServer* service. On the other hand, *MSExchangeMTA* and *MSExchangeIS* rely on the *MSExchangeSA* service. The association between these services is clearly there, but it works in the other direction. Later in this section we see how the dependency direction is set when we examine the association classes. Note that it is possible to see these associations with the **WMI Object Browser**. We do not use this tool here because the **WMI Object Browser** is not designed to review the class definitions; it is designed to examine the class instances only.

How does WMI make these links? By simply using the association mechanism defined in the CIM classes definitions. To understand this, we must go back to the *Win32_Service* class definition and, using the **WMI CIM Studio**, look at the associations made in the class definition. **WMI**

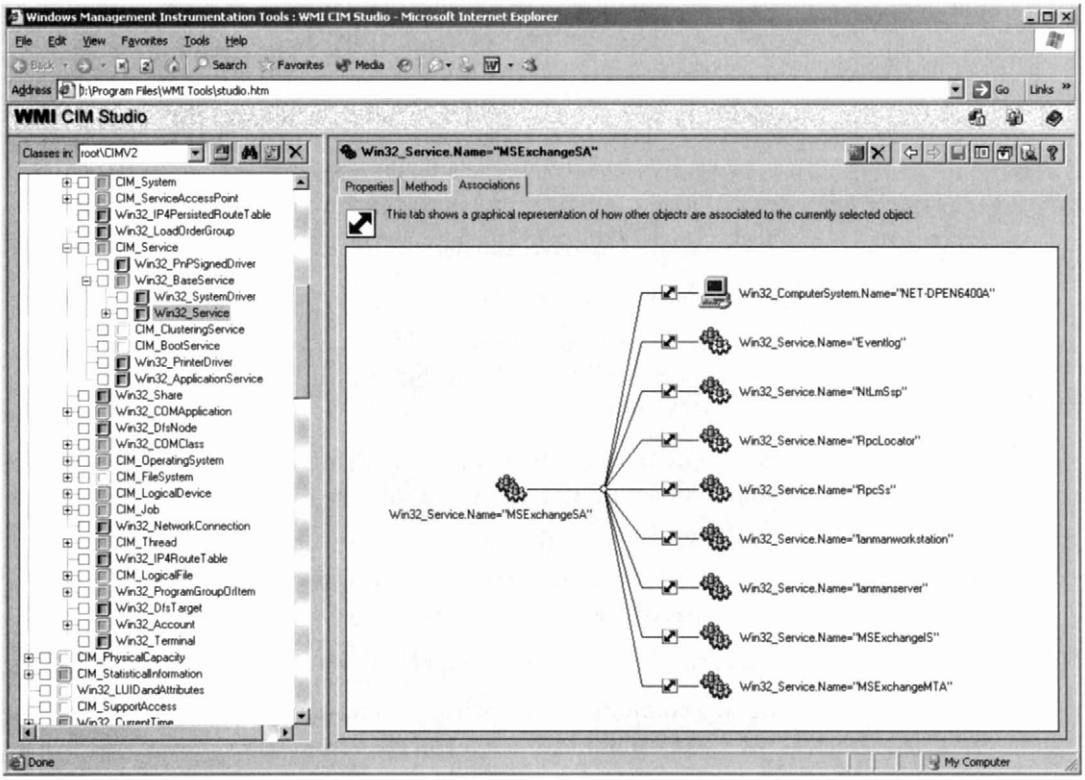

Figure 2.33 *Viewing the Microsoft Exchange 2000 System Attendant service dependencies with WMI CIM Studio.*

CIM Studio shows three dependencies from the *Win32_Service* class, as shown in Figure 2.34.

We see that the *Win32_Service* class is associated to *Win32_WMISetting*, *Win32_ServiceSpecification*, and *Win32_ComputerSystem* classes. The association is made with three association classes (represented in Figure 2.34

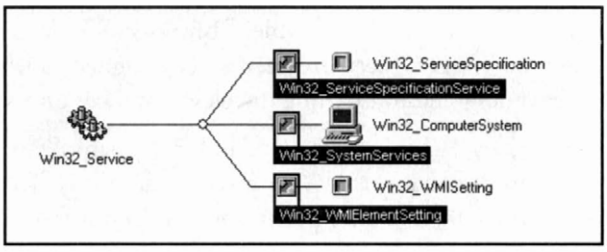

Figure 2.34 *The associated classes and associations classes of the Win32_Service class.*

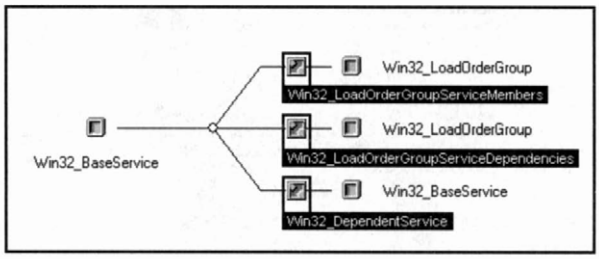

Figure 2.35
The associated classes and associations classes of the Win32_BaseService class.

with a circled icon with its name in a black box), which are *Win32_WMIElementSetting*, *Win32_ServiceSpecificationService*, and *Win32_SystemServices* classes.

Now, if we examine the parent class (the class from which the *Win32_Service* is derived), we see that *Win32_BaseService* also has associations with specific classes (see Figure 2.35). These are *Win32_LoadOrderGroup* and *Win32_BaseService* classes. In the same way as before, these associations are made with some association classes that include: *Win32_LoadOrderGroupServiceMembers*, *Win32_DependentService*, and *Win32_LoadOrderGroupServiceDependencies* classes. You will notice in Figure 2.35 that the *Win32_BaseService* class is associated with itself. This makes sense, since a service can rely on another service.

Because these associations are inheritable by the subclasses, the *Win32_Service* class also inherits these associations. Why are these associations inheritable? To give the answer, we must examine the association class qualifiers. Let's view the association class called *Win32_DependentService* by performing a search with **WMI CIM Studio** to locate the class. Once found, select the class and right-click the right pane showing the properties. Then select the Object qualifiers. You should obtain what is shown in Figure 2.36. This figure shows the list of qualifiers used to define the class type and behavior. In the list you will find the provider qualifier stating that the class is working with the Win32 provider (*CIMWin32*); you will also see a qualifier stating that the class is dynamic because any class instance is provided by the provider itself and not stored statically in the CIM repository. In that list, the association qualifier is presented at the top and set to True.

Figure 2.36 shows that the association qualifier is checked to replicate to instances and derived classes. This means that every instance of this association class or every class derived from this association class will also inherit these associations.

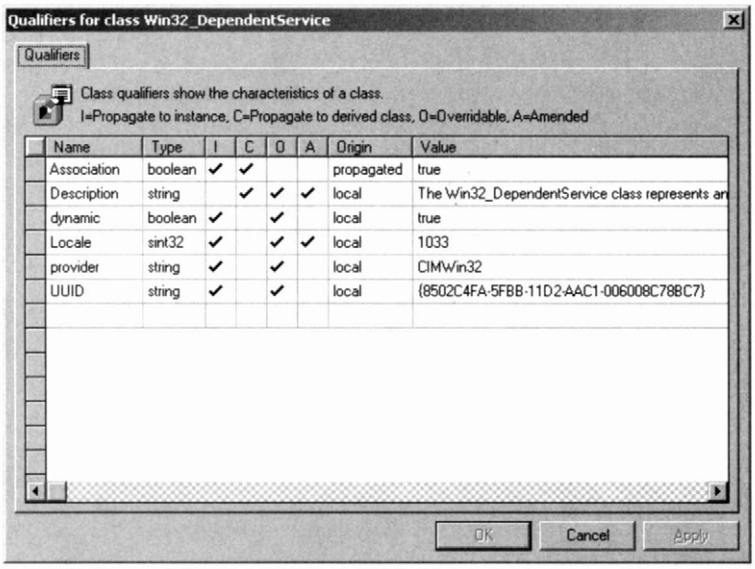

Figure 2.36 *The Win32_DependentService association class qualifiers.*

In summary, the *Win32_Service* class is associated by inheritance or directly with the following classes:

- *Win32_WMISettings* directly from *Win32_Service* class.
- *Win32_ServiceSpecification* directly from *Win32_Service* class.
- *Win32_ComputerSystem* directly from *Win32_Service* class.
- *Win32_LoadOrderGroup* by inheritance from *Win32_BaseService* class.
- *Win32_BaseService* by inheritance from *Win32_BaseService* class with the following association classes:
  - *Win32_WMIElementSetting* directly from *Win32_Service* class.
  - *Win32_ServiceSpecificationService* directly from *Win32_Service* class.
  - *Win32_SystemServices* directly from *Win32_Service* class.
- *Win32_LoadOrderGroupServiceMembers* by inheritance from *Win32_BaseService* class.
- *Win32_DependentService* by inheritance from *Win32_BaseService* class.
- *Win32_LoadOrderGroupServiceDependencies* by inheritance from *Win32_BaseService* class.

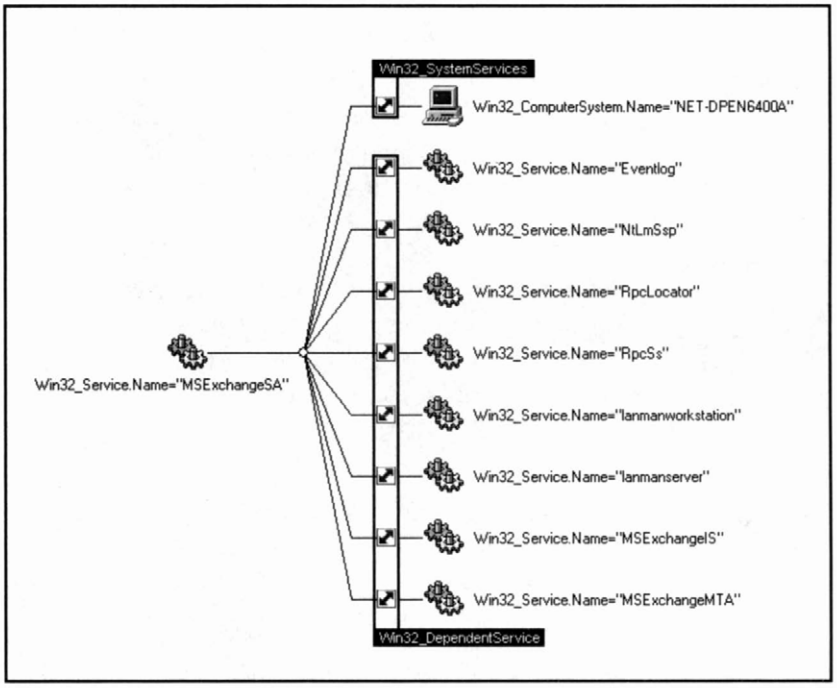

Figure 2.37 *Win32_Service MSExchangeSA instance associated instances and associations instances.*

So, as in the beginning of this section, when we looked at the associated instances of the *MSExchangeSA* service instance, we found associated instances made from only two different classes (see Figure 2.37):

1. One *Win32_ComputerSystem* instance coming directly from the association of the *Win32_Service* class
2. Eight *Win32_Service* instances coming from the inheritance of *Win32_BaseService* class.

Why two class instances and not all the associated class instances? It is very simple: There is no other existing instance of the associated classes other than the two listed with the *MSExchangeSA* instance. In short, we can say that the *MSExchangeSA* relies only on some instances of the *Win32_Service* class and on one instance of the *Win32_ComputerSystem* class. Now, you can take another service such, as Windows Management Instrumentation (WinMgmt), and you will see that there are differences. The service dependency is different. You see that there is a new instance associated with it that comes from the *Win32_WMISetting* class. This class represents nothing more than the WMI settings configurable by an administrator via the MMC shown in Figure 2.20. In this case, it makes sense to

have such a class instance associated with the **WinMgmt.exe** service instance.

Now, how does the association class work? If we consider the *Win32_DependentService* association class properties, you will see that the class defines three properties, two of which are used as a key to uniquely identify the associated classes. These two keys are *Antecedent* and *Dependent*. Both point to a *Win32_Service* class as stated by their type "ref:Win32_Service." As these two keys are properties using a "ref:" type in an association, they are also references. If you remember, we mentioned in this chapter that the properties of an association are the references. This is why when we talked earlier about associations, references, and domain, we said that the references have a domain limited to the association.

In Figure 2.38, the *Antecedent* reference points to a *Win32_Service* instance that must be started in order to start the *MSExchangeSA* service. The *Dependent* reference points to a *Win32_Service* instance that can be started if the *MSExchangeSA* service is started. The reference type determines the type of dependency that the class has with its associated class. As instances come from classes, it is exactly the same for the instances, which is why the *MSExchangeMTA* and *MSExchangeIS* are listed in the associated instance list in Figure 2.23. The *MSExchangeMTA* and *MSExchangeIS* depend on the *MSExchangeSA* to start. In the same window, if you move

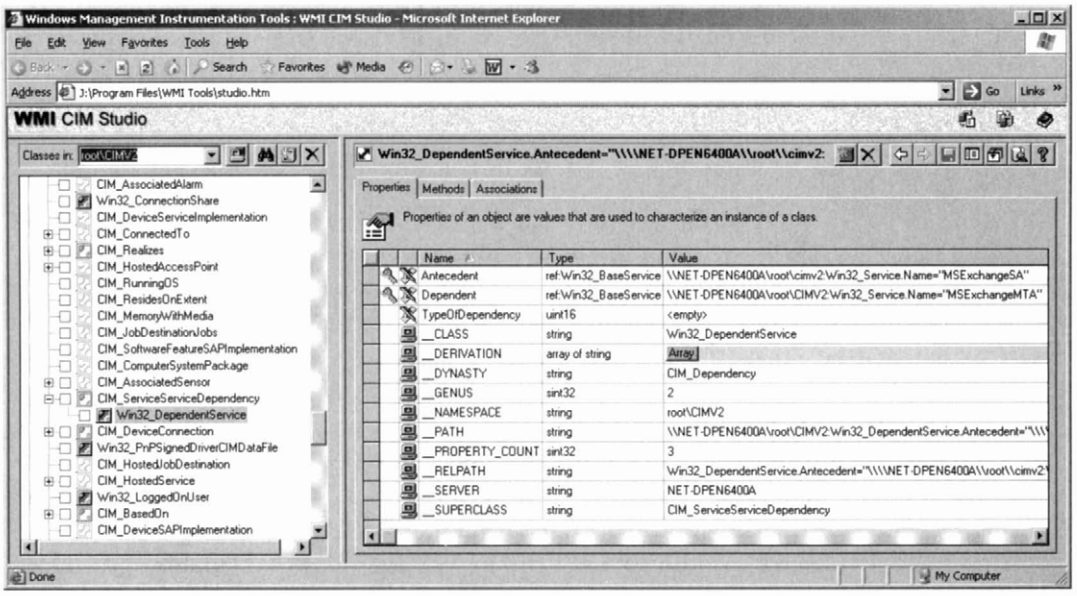

Figure 2.38 *The Win32_DependentService association class with its reference definitions.*

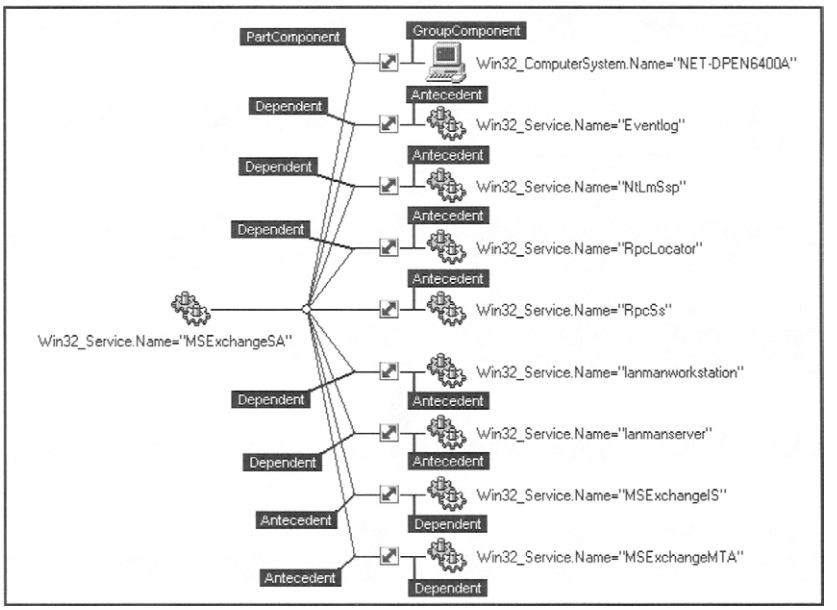

Figure 2.39 *The association inversion based on the relationships between instances.*

the mouse over the lines linking the association instances with the class instances, the reference name appears on top. Basically, you see a view of the names of references similar to that shown in Figure 2.39.

Note the inversion of the Antecedent and Dependent references, for example, if you compare the *MSExchangeMTA* service and the *LanManServer* service. The definitions of the references that point to the computer object where the *MSExchangeMTA* service resides come from the association class *Win32_SystemServices*. Even if the classes are different, the logic is exactly the same for all classes, class instances, and associations exposed by WMI.

Based on this discovery, it is easy to imagine that we can take advantage of these associations to retrieve the Windows service dependencies without reading the content of the registry key.

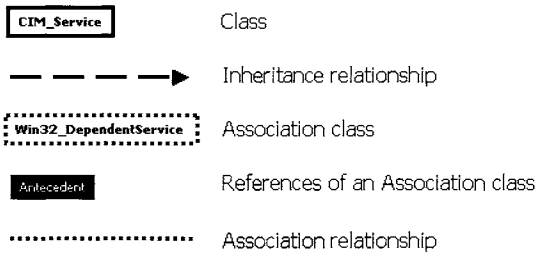

Figure 2.40 *Legend for Figure 2.41.*

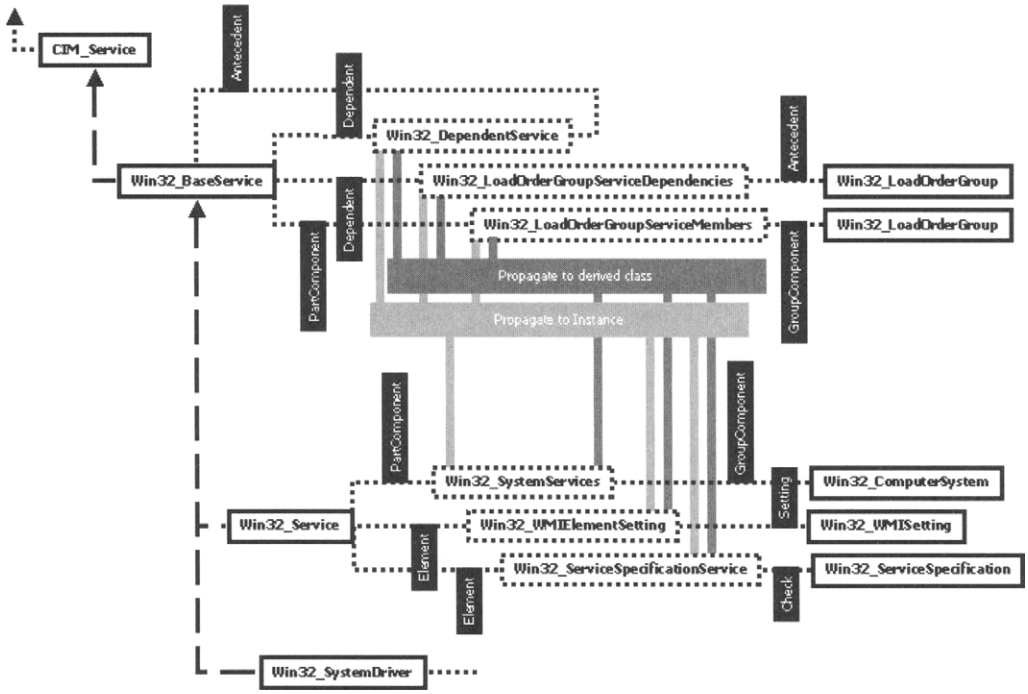

Figure 2.41 *A schematic representation of the various links between the service classes.*

To avoid any confusion, it is important to keep in mind that the associations that exist between the superclasses are inherited by the subclasses. Figures 2.40 and 2.41 show a logical representation of these relations. However, this representation is not a formal representation of these relationships since it is not an UML representation. The only purpose of this representation is to illustrate explicitly all logical links that exist between the various classes (superclasses and subclasses). The Windows services dependency of the *MSExchangeSA* service is a nice example that is reused throughout subsequent chapters.

2.9.2 An event consumer application without scripting

While talking about the providers and the consumers, we saw that WMI comes with its own set of ready-to-use event consumers. It is possible, with the help of a MOF file, to associate an event that comes from a particular provider with a permanent event consumer. We know that the *WMI Event Viewer* is a permanent event consumer. Based on this, it is possible to create a simple alerting system using the various WMI Microsoft components and tools. This can be achieved without writing a single line of code and by sim-

ply using the WMI architecture and its implementation. The alert system monitors any changes made to the Windows services and receives an alert when such a change occurs. If a startup state of one service is changed or if one service is started or stopped, WMI will trigger an event forwarded to the event consumer. Of course, this exercise introduces the notion of events that we examine later in Chapter 6, but let's take a shortcut to illustrate the concepts we have covered in this chapter.

When we used the **WBEMTEST.exe** tool, we used a query string to list the Windows Services. This query string was

```
Select * From Win32_Service
```

By adapting this query string to the event notification mechanism, we can formulate the following query:

```
Select * From __InstanceModificationEvent Within 10 Where
TargetInstance ISA 'Win32_Service'
```

Basically, both queries retrieve the same information, except that the latter retrieves the information only when a change occurs in an instance of the *Win32_Service* class. Simply said, each time a modification is made to a Windows Sservice, the query finds a match and returns all of the properties of the modified service. The **Within 10** statement informs WMI to perform a polling every 10 seconds to detect changes made. The presence of this parameter is determined by the nature of the WMI provider.

To link an event detected by a WMI provider to a WMI permanent event consumer, there are three important things to define:

1. **The event consumer instance**: When talking about the *WMI Event Viewer*, we saw a MOF file that defines the class representing the *WMI Event Viewer* template. Now, the idea is to create an instance of the permanent event consumer class in order to get a real permanent event consumer. A real permanent event consumer has a name and a set of parameters associated with it. This means that we will use the permanent event consumer class of the *WMI Event Viewer* as a template to create the real *WMI Event Viewer* object with a set of explicitly defined parameters. Don't forget that the *WMI Event Viewer* must be registered in the CIM repository; the following command line will perform the registration:

```
C:\>mofcomp "%ProgramFiles%\WMI Tools\EViewer.mof"
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: EViewer.Mof
MOF file has been successfully parsed
```

```
Storing data in the repository...
Done!
```

2. **The event filter**: Because the idea is to receive a notification for every change made to the Windows services, the event filter will filter the events that match the submitted query.

3. **The link between the event filter and the permanent event consumer**: Its purpose is to link the event filter instance with the permanent event consumer instance. This element is typically the case of an association. So, every event that matches the filter is sent to the WMI Event Viewer.

There are several ways to achieve this. First, we can use a script to create the necessary definition in the CIM repository. Because we don't have the required WMI scripting background yet, let's skip this method and save it for later. A second method is to use the WMI Event Registration tool. Using this tool implies coding, through the user interface, of all the necessary parameters. Although possible, this is not the easiest way to perform this task. It is best to use the WMI Event Registration tool to modify an existing registration. The last method, and also the easiest, is to use a MOF file similar to that shown in Sample 2.2.

Sample 2.2 *A MOF file to associate the permanent WMI Event Viewer consumer and any change notification coming from the Win32_Service instances (EventViewerConsumerInstanceReg.mof)*

```
1:// ---------------------------------
2://   Sample for EventViewerConsumer
3://---------------------------------
4:instance of __EventFilter as $ef
5:{
6:  Name = "FilterForWin32_Services";
7:  Query = "SELECT * FROM __InstanceModificationEvent WITHIN 10 Where "
8:          "TargetInstance ISA 'Win32_Service'";
9:  QueryLanguage = "WQL";
10:};
11:
12:// create the consumer
13:instance of EventViewerConsumer as $eventvwrec
14:{
15:  Name = "EventViewerForSvc";
16:  Severity = 2;
17:  Description = "Event viewer consumer associated to any Win32_Service event";
18:};
19:
20:// bind the filter and the consumer
```

```
21:instance of __FilterToConsumerBinding
22:{
23:  Filter = $ef;
24:  Consumer = $eventvwrec;
25:};
```

As previously mentioned, the MOF file contains three sections. The first section defines the event filter (lines 4 to 10). Besides the name given to the event filter (line 6) we recognize the query acting as a filter (lines 7 and 8). The next section defines the instance of the permanent event consumer (lines 12 to 18). This instance is created from the *WMI Event Viewer* class. This is why line 13 contains a statement with the word *EventViewerConsumer*, as this is the class name of the permanent event consumer registered in the CIM repository (see the MOF file presented in Section 2.8.5.5). We also see that some properties are set, such as the name (mandatory as the name is a key of the class), severity, and a description property. Table 2.11 summarizes the parameters of the *WMI Event Viewer* event consumer.

Once the filter event instance and the permanent event consumer instance are defined, the MOF file associates these two instances together with the help of third instance made from the *_FilterToConsumerBinding* system class. This class is simply an association class linking the consumer instance with its filter instance. You will notice that the association is made by referencing an alias *$ef* (for the event filter at lines 4 and 23) and *$eventvwrec* (for the *Event Viewer* event consumer at lines 13 and 24). By compiling the MOF file, these three instances will be loaded in the CIM

Table 2.11 *The WMI Event Viewer Event Consumer Class*

Name	AccessType: Read
	Qualifier: Key
	String the unique identifier of the event consumer
Severity	AccessType: Read
	Qualifier: Template
	0 Error
	1 Warning
	2 Information
Description	AccessType: Read
	Qualifier: Template
	String describing the event

repository. This will also inform CIMOM about the action to perform. Let's compile the MOF file with the following command line:

```
C:\>mofcomp -n:Root\CIMv2 EventViewerConsumerInstanceReg.Mof
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: EventViewerConsumerInstanceReg.Mof
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

Before loading Sample 2.2 in the CIM repository, the Eviewer.mof (see Section 2.8.5.5) must be loaded first. The Eviewer.mof registers the WMI Event Viewer COM components in the **Root\CIMv2** namespace of the CIM repository. Next, when registering Sample 2.2, it is important to specify the –N switch to force the selected namespace to **Root\CIMV2**; otherwise the default selected namespace will be **Root\Default**. In this case, the compilation will not work because the WMI event consumer class and the *Win32_Service* are not registered in the **Root\Default** namespace of the CIM repository. Both the *EventViewerConsumer* class and the *Win32_Service* class are registered by default in the **Root\CIMV2** namespace. WQL does not support cross-namespace queries or associations. You cannot query for instances of a specified class residing in all the namespaces on the target computer. Furthermore, you cannot associate two objects across a namespace boundary and retrieve or query for the associations. All items must reside beneath the same namespace. However, there is limited support for cross-namespace bindings in that all statically defined (repository-based) objects can reference each other across namespace boundaries. Once providers become involved, there is no such support and queries are always executed with reference to a particular namespace.

Once completed, you can use the WMI Event Registration tool to check the event filter instance and the WMI permanent event consumer instance, as shown in Figure 2.42.

If a modification must be made after the registration, it is always possible to modify the MOF file and perform a new compilation (in such a case, make sure that you don't change the instance names to address the existing instances). Alternatively, you can use the WMI Event Registration tool to perform the changes. If you take a look at these instances with the WMI Event Registration tool, you should start to understand why it is easier to create these three classes with a MOF file.

Do we need to do something else before getting notifications? No! Everything is ready! In a previous paragraph we saw that the WMI event

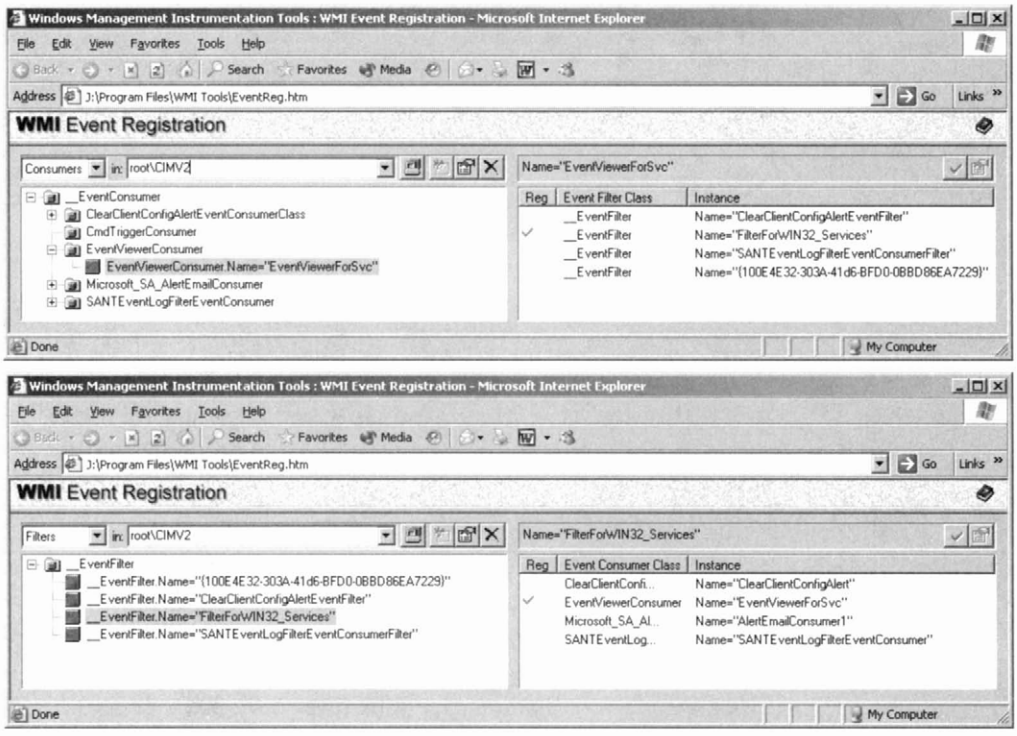

Figure 2.42 *Viewing the WMI Event Viewer instance and the Event Filter instance with the WMI Event Registration tool.*

provider application is a permanent event consumer; this implies that WMI will start the application if an event related to a *Win32_Service* instance occurs. Let's test this by selecting a Windows service that you can stop or start (i.e., take the SNMP service as it is not a key service to run Windows). If everything is fine, the WMI Event Viewer should start, and the screen shown in Figure 2.43 will appear.

Each time something changes in the *Win32_Service*, a similar event will be visible in the WMI Event Viewer. If you click twice on the event, you will see the properties of the event (see Figure 2.44).

Among the various properties, two are particularly interesting: the *PreviousInstance* and the *TargetInstance* properties. The *PreviousInstance* property represents the *Win32_Service* instance as it was before the change occurred. Therefore, if the action triggering the change notification was a service start, this implies that the service was stopped; the object embedded in this property will be in a stopped state. The *TargetInstance* property shows the instance in its current state. So, in the case of our example, the instance

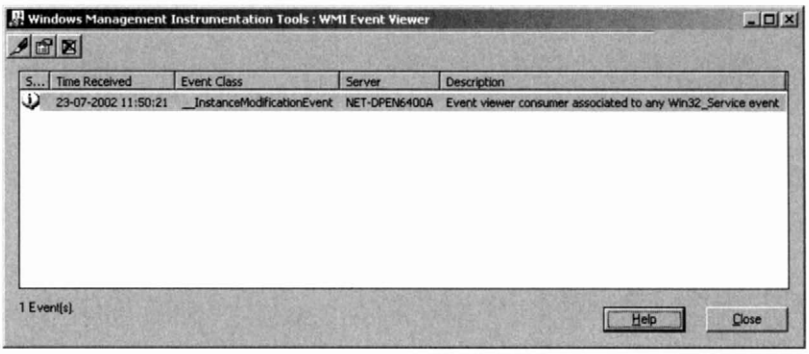

Figure 2.43 *The WMI Event Viewer receiving the event.*

embedded in the *TargetInstance* property will be in a started state. Besides the state of the service, both instances show the properties of the Windows service.

It is nice to get a popup window with the *WMI Event Viewer*, but it would be nicer to receive an e-mail each time a change occurs. As previously shown, WMI provides a permanent *SMTP* event consumer. This means that we can use the same principle as the *WMI Event Viewer* consumer, but in this case, we must create an instance of the *SMTP* event consumer with the necessary SMTP parameters. When we previously discussed the **MOFCOMP.exe** tool, we showed the MOF file that defines the permanent *SMTP* event consumer in the CIM repository (see Section 2.8.5.1). This MOF file shows the required properties for the permanent *SMTP* event consumer. Note that the registration is made during the WMI installation

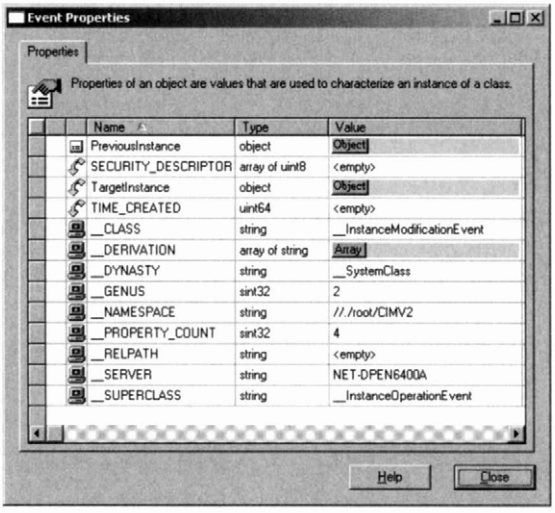

Figure 2.44 *The event properties as visible in the WMI Event Viewer.*

in the **Root\Subscription** namespace, but the *Win32_Service* class is available only in the **Root\CIMv2** namespace. Since we cannot work across two different namespaces, it is mandatory to register the permanent *SMTP* event consumer in the **Root\CIMv2** namespace by using the following command line:

```
C:\>mofcomp -N:Root\CIMv2 %SystemRoot%\System32\Wbem\SMTPCons.Mof
Microsoft (R) 32-bit MOF Compiler Version 5.1.3590.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: J:\WINDOWS\System32\Wbem\SMTPCons.Mof
MOF file has been successfully parsed
Storing data in the repository...
```

As the permanent *SMTP* event consumer class is now available in the **Root\CIMv2** namespace, it is now possible to associate a permanent *SMTP* event consumer instance with an event filter instance (as we made before for the *WMI Event Viewer* consumer). The MOF file performing this definition is shown in Sample 2.3.

Sample 2.3 *A MOF file to associate the permanent SMTP event consumer and any change notification coming from the Win32_Service instances (SMTPConsumerInstanceReg.mof)*

```
1:// --------------------------------
2://   Sample for SMTPEventConsumer
3://--------------------------------
4:instance of __EventFilter as $ef
5:{
6:  Name = "FilterForWin32_Services";
7:  Query = "SELECT * FROM __InstanceModificationEvent WITHIN 10 Where "
8:          "TargetInstance ISA 'Win32_Service'";
9:  QueryLanguage = "WQL";
10:};
11:
12:// create the consumer
13:instance of SMTPEventConsumer as $smtpec
14:{
15:  Name = " SMTPForSvc";
16:  Subject = "Service %TargetInstance.DisplayName% is %TargetInstance.State%";
17:  Message = "Service %TargetInstance.DisplayName% (%TargetInstance.Name%) "
18:            "is %TargetInstance.State%.\n"
19:            "Startup mode is %TargetInstance.StartMode%.";
20:  FromLine = "WMISystem@LissWare.NET";
21:  ToLine = "Alain.Lissoir@LissWare.NET";
22:  SMTPServer = "smtp.LissWare.NET";
23:};
24:
25:// bind the filter and the consumer
26:instance of __FilterToConsumerBinding
27:{
28:  Filter = $ef;
29:  Consumer = $smtpec;
30:};
```

As we can see, the MOF file redefines the event filter. As this filter is the same as the one used with the *WMI Event Viewer*, it is not necessary to redefine it here. We can use the MOF file listed in Sample 2.4.

Sample 2.4 *A MOF file to associate the permanent SMTP event consumer and an existing event filter (SMTPConsumerInstanceReg2.mof)*

```
 1:// ---------------------------------
 2://   Sample for SMTPEventConsumer
 3://---------------------------------
 4:
 5:// create the consumer
 6:instance of SMTPEventConsumer as $smtpec
 7:{
 8:  Name = "SMTPForSvc";
 9:  Subject = "Service %TargetInstance.DisplayName% is %TargetInstance.State%";
10:  Message = "Service %TargetInstance.DisplayName% (%TargetInstance.Name%) "
11:            "is %TargetInstance.State%.\n"
12:            "Startup mode is %TargetInstance.StartMode%.";
13:  FromLine = "WMISystem@LissWare.NET";
14:  ToLine = "Alain.Lissoir@LissWare.NET";
15:  SMTPServer = "smtp.LissWare.NET";
16:};
17:
18:// bind the filter and the consumer
19:instance of __FilterToConsumerBinding
20:{
21:  Filter = "FilterForWin32_Services";
22:  Consumer = $smtpec;
23:};
```

Only the definition of the event filter is missing. The association class (lines 19 to 23) references the existing event filter made for the *WMI Event Viewer* (line 21). In any case, the MOF definition for the permanent *SMTP* event consumer instance will not disconnect the *WMI Event Viewer* from the event filter, because the association class instance *__FilterToConsumerBinding* uses two keys to be differentiated from the association class instance made for the *WMI Event Viewer*. These two keys combine the name of the event filter (which is the same) and the name of the event consumer (which is different). As the combination of the two keys is not the same, the previous association instance is not overwritten as shown in Figure 2.45. Make sure that both consumers are registered as shown in the figure. The two keys are defined in the *__FilterToConsumerBinding* system class (visible with the **WMI CIM Studio**).

Besides the association aspect, the event consumer is different and requires specific parameters related to SMTP. The first thing needed is an SMTP server available in your network. This server is mandatory. Any of

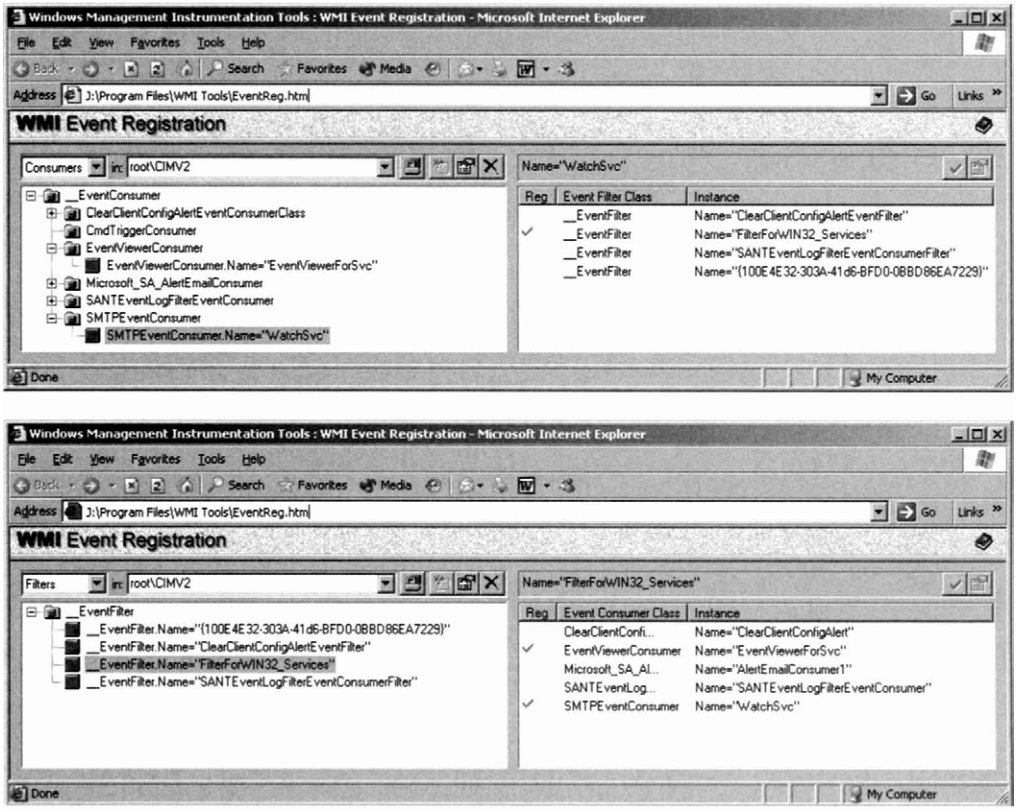

Figure 2.45 *The WMI event filter registration for two different permanent event consumers.*

the To, Cc, and Bcc parameters can be null, but they may not all be null. Next, the server name is set at line 15; the target recipient is set at line 14. The permanent *SMTP* event consumer does not support attachments in mail. Table 2.12 summarizes the available parameters for the permanent *SMTP* event consumer. In addition to these parameters, make sure that the SMTP server relaying the message is well configured. If the SMTP server requires an authentication mechanism, this will fail as the permanent *SMTP* event consumer does not provide parameters to perform an authentication mechanism. Make sure that the SMTP server allows anonymous access. Compile the MOF file as follows:

```
C:\>MOFCOMP -N:Root\CIMV2 SMTPConsumerInstanceReg.mof
```

or, if the filter instance already exists:

```
C:\>MOFCOMP -N:Root\CIMV2 SMTPConsumerInstanceReg2.mof
```

Table 2.12 *The SMTP Event Consumer Class*

Name	Access type: Read/write Qualifiers: Key String the unique identifier of the event consumer.
Subject	Access type: Read/write Qualifiers: Template Template string containing the subject of the e-mail message.
Message	Access type: Read/write Qualifiers: Template Template string containing the body of the e-mail message.
ToLine	Access type: Read/write Qualifiers: Template String containing a comma or semicolon-separated list of e-mail addresses where the message is to be sent.
FromLine	Access type: Read-only Qualifiers: Template From line of the e-mail message. If NULL, a From line is constructed of the form WinMgmt@MachineName.
ReplyToLine	Access type: Read-only Qualifiers: Template Reply-to line of the e-mail message. If NULL, no reply-to line is used.
SMTPServer	Access type: Read/write Qualifiers: Not null String that names the SMTP server through which the e-mail is to be sent. Permissible names are an IP address, or a DNS or NetBIOS name. This property cannot be NULL.
CcLine	Access type: Read/write Qualifiers: Template String containing a semicolon-separated list of addresses to which the message is sent as a carbon copy.
BccLine	Access type: Read/write Qualifiers: Template String containing a semicolon-separated list of addresses to which the message is sent as a blind carbon copy.
HeaderFields	Access type: Read-only String array of header fields that are inserted into the e-mail message without interpretation.

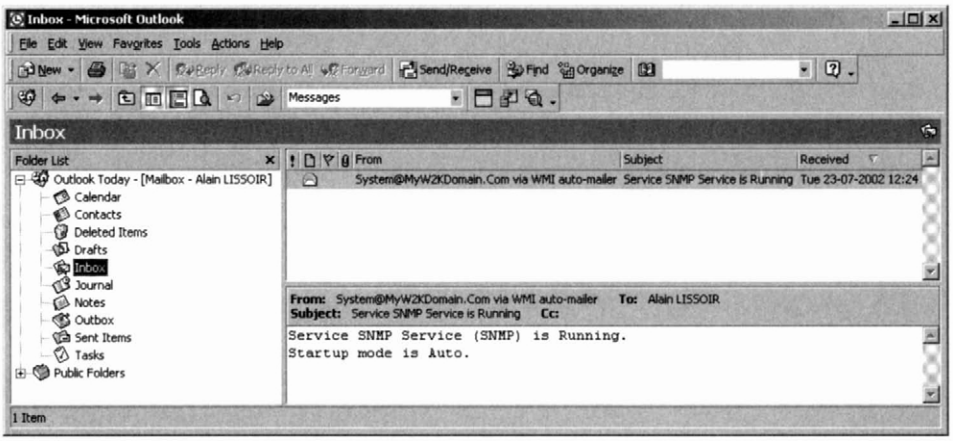

Figure 2.46 *The received mail from the permanent SMTP event consumer.*

Once completed, the only way to test the new CIM instance definition for the *SMTP* event consumer is to perform a change on a Windows service (by stopping or starting a service or modifying the startup mode of a service). When a change occurs the target recipient will receive an e-mail similar to that shown in Figure 2.46.

2.10 Summary

This chapter explained the basic foundation of WMI. Some of the WMI features available were used to illustrate the possibilities offered by this management instrumentation without programming. A very important point is that you should be trained with the WMI tools provided by Microsoft because each time it is necessary to work with or gather information about classes, instances, events, providers or consumers, these tools are extremely useful. When we examine the scripting techniques on top of WMI, these tools will also be a complementary aid to the development process.

Gathering knowledge about the CIM repository and its object model is a long process. The most difficult part is to distinguish clearly the difference between the CIM elements and how they interact with one another. Understanding the difference between a class and an instance is probably basic, but it is also very important! Under CIM everything is defined in the form of classes and instances; therefore, the WMI tools delivered by Microsoft with the standard WMI installation or as a separate download from the Web are particularly helpful. We encourage you to play with these tools and examine how various elements are defined.

As the CIM repository is a specialized database modeling real-world manageable entities, with the help of the tools (e.g., WMIC, WBEMTEST, WBEMDUMP, and **WMI CIM Studio**) and the specialized WMI query language (WQL), it is possible to perform various queries to retrieve the information stored in the CIM repository. This language is an important key player when creating scripts that monitor real-world manageable entities. The next chapter delves into the WQL world.

2.11 Useful Internet URLs

Windows Management Instrumentation (WMI) CORE 1.5 (Windows 95/98/NT 4.0):

```
http://www.microsoft.com/downloads/release.asp?releaseid=18490
```

Windows Management Instrumentation (WMI) SDK 1.5 (Windows NT 4.0 and 2000):

```
http://www.microsoft.com/downloads/release.asp?releaseid=19157
```

Platform SDK:

```
http://www.microsoft.com/msdownload/platformsdk/sdkupdate
```

WMI Tools:

```
http://www.microsoft.com/downloads/release.asp?ReleaseID=40804
```

Scriptomatic:

```
http://www.microsoft.com/downloads/release.asp?ReleaseID=41528
```