3

The WMI Query Language

3.1 Objective

In the previous chapter, when we performed basic monitoring using the *SMTP* event consumer, we used a WQL query to define and filter the event. When we used the **WMI CIM Studio** to browse the content of the CIM repository, we did not use a WQL query to retrieve the information, even though **WMI CIM Studio** provides the ability to perform WQL queries to retrieve CIM information. In the same way, **WBEMTEST.exe** and **WBEMDump.exe** are also capable of performing WQL queries. Mastering WQL is very important because it can be used in different places. When working with the scripting interfaces and the WMI events, it is impossible to avoid its usage because a WQL query is the required primary input. This is why, in this chapter, we do not dive immediately into the scripting technique, and instead we focus on WQL. After completion of this chapter, we will be ready to start exploring the WMI scripting API.

Important notice: At this stage, it is really important to understand the relationship between the Windows services as described in Chapter 2 (Section 2.9.1). Don't hesitate to compare the result of WQL queries with what is available from **WMI CIM Studio**. It will be very important for you to practice the queries during this learning phase. It is easy to get lost and confused without practicing!

3.2 The WMI query language

As the purpose of the CIM repository is to create a data model that represents real-world manageable entities, it is important to have a powerful technique to retrieve the information stored in the repository. Besides the

simple browsing or the exploration of WMI possibilities, WMI also comes with a specific query language, which is an alternate method to retrieve information from the CIM repository. The query language is a subset of the American National Standards Institute Structured Query Language (ANSI SQL) and is called WMI Query Language (WQL). However, it differs from the standard SQL in that it retrieves from classes rather than tables and returns WMI classes or instances rather than rows. This language is an important tool addition to WMI because it facilitates data retrieval and can be used in various ways from many different tools. To suit WMI's needs, WQL has some semantic changes when compared with SQL. Unlike SQL, WQL is a read-only language. This means that WQL does not allow operations to modify, insert, or delete information from the CIM repository. As briefly mentioned in the previous chapter, WQL is mainly used to perform the following:

- **Data queries:** to retrieve instances of classes and information about instance associations
- **Schema queries:** to retrieve class definitions and information about schema associations
- **Event queries:** to be used by temporary consumers or permanent consumers. Basically, these queries are used to filter WMI events

All WQL queries use reserved keywords. Table 3.1 lists the supported WQL keywords with the context in which they can be used.

Table 3.1 *WQL Keywords*

	Query Type			
Keywords	**Data**	**Schema**	**Event**	**Description**
__CLASS	*	*		References the class of the object in the query. This keyword is available in Windows 2000 and later.
AND	*(1)		*	Combines two Boolean expressions and returns TRUE when both of the expressions are TRUE.
ASSOCIATORS OF	*	*		Retrieves all instances that are associated with a source instance. Use this statement with schema queries and data queries.
BY			*	Indicates which properties to use when grouping. Use this clause with the GROUP clause.

(1) Not usable with the "Associators of" and the "References of" statements.

Table 3.1 *WQL Keywords (continued)*

Keywords	Query Type			Description
	Data	Schema	Event	
FALSE	*	*	*	Boolean operator that evaluates to 0.
FROM	*	*	*	Specifies the classes that contain the properties listed in a SELECT statement.
GROUP			*	Causes WMI to generate a single notification to represent a group of events. Use this clause with event queries.
HAVING			*	Filters the events that are received during the grouping interval specified in the WITHIN clause.
IS			*	Comparison operator used with NOT and NULL. The syntax for this statement is IS [NOT] NULL where NOT is optional.
ISA	*	*	*	Operator that applies a query to the subclasses of the specified class. For more information, see ISA Operator for Event Queries, ISA Operator for Data Queries, and ISA Operator for Schema Queries.
KEYSONLY	*			Used in REFERENCES OF and ASSOCIATORS OF queries so that the resulting instances are only populated with the keys of the instances, thereby reducing the overhead of the call. This keyword is available in Windows XP and later.
LIKE	*			Operator that determines whether a given character string matches a specified pattern
NOT	*		*	Comparison operator that can be used in any WQL SELECT query.
NULL	*		*	Indicates that an object has no explicitly assigned value. NULL is not equivalent to zero or blank.
OR	*(1)		*	Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators.
REFERENCES OF	*	*		Retrieves all association instances that refer to a particular source instance. Use this statement with schema and data queries. The REFERENCES OF statement is similar to the ASSOCIATORS OF statement. However, rather than retrieving endpoint instances, it retrieves the intervening association instances.

(1) Not usable with the "Associators of" and the "References of" statements.

Table 3.1 *WQL Keywords (continued)*

Keywords	Query Type			Description
	Data	Schema	Event	
SELECT	*	*	*	Specifies the properties that will be used in a query. For more information, see SELECT Statement for Data Queries, SELECT Statement for Event Queries, or SELECT Statement for Schema Queries.
TRUE	*		*	Boolean operator that evaluates to –1.
WHERE	*	*	*	Narrows the scope of a data, event, or schema query.
WITHIN			*	Specifies either a polling interval or a grouping interval. Use this clause with event queries.

(1) Not usable with the "Associators of" and the "References of" statements.

Besides the reserved keywords, WQL also uses operators. Some of the keywords can be considered as operators. Table 3.2 contains a list of the operators that can be used in a WQL query.

As we examine the different types of queries, we revisit each keyword and operator usage. For now, let's start with the first one: the data queries.

Table 3.2 *WQL Operators*

Operator	Definition
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!= or <>	Not equal to
IS	Test if a constant is NULL
IS NOT	Test if a constant is NOT NULL
ISA	For Data and Event queries, tests embedded objects for a class hierarchy. For Schema Queries, Newly-created and existing subclasses of the requested class are automatically included in the result set.

3.2.1 Data queries

In the previous chapter, we saw that instances contain information about real-world manageable entities. The data quey's purpose is to retrieve information from these real-world manageable entities. We saw that some classes are associated together, which implies that instances of these classes are also associated together. In other words, it is possible to retrieve the associations and the references related to an instance of a class.

3.2.1.1 Retrieving instances of classes

As the philosophy of this book is to learn by practice, let's take the same easy sample used in the previous chapter:

```
Select * From Win32_Service
```

The **SELECT** statement is one of three statements available to perform a data query. We revisit the other two later in this section. To perform this query, we can use any tool offering WQL query capabilities. Instead of using **WBEMTEST.exe** or **WMI CIM Studio** as before, we use **WBEMDump.exe**. As a reminder, **WBEMDump.exe** is an application sample that comes with the Platform SDK (see Chapter 2). To execute this sample query with **WBEMDump.exe**, use the following command line:

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Select * From Win32_Service"
```

The **/E** enables to display the system classes and the system properties. The **/Q** enables the WMI query features of **WBEMDump.exe**. Next, the query is executed in the **Root\CIMv2** namespace, and WQL is used to perform the query (currently, this is the only language supported by WMI). The last parameter is the WQL query itself. Once executed, the query output is similar to that shown in Sample 3.1. The output has been reduced to minimize the amount of data, as this query displays all information available from all *Win32_Service* instances. In the system properties, we can see the *__CLASS* property containing the name of the class. This information shows the class name of the instance.

Sample 3.1 *Using the SELECT statement to get all properties of all instances of a given class*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Select * From Win32_Service"
(WQL) Select * From Win32_Service
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        __RELPATH (CIM_STRING/)  = "Win32_Service.Name="Alerter""
        ...
        __SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
        AcceptPause (CIM_BOOLEAN/boolean)  = FALSE
```

```
AcceptStop (CIM_BOOLEAN/boolean)  = TRUE
Caption (CIM_STRING/string)  = "Alerter"
CheckPoint (CIM_UINT32/uint32)  = 0 (0x0)
CreationClassName (CIM_STRING/string)  = "Win32_Service"
Description (CIM_STRING/string)  = "Alerter"
DesktopInteract (CIM_BOOLEAN/boolean)  = FALSE
Displayname (CIM_STRING/string)  = "Alerter"
ErrorControl (CIM_STRING/string)  = "Normal"
ExitCode (CIM_UINT32/uint32)  = 0 (0x0)
InstallDate (CIM_DATETIME/datetime)  = <null>
Name (CIM_STRING/string)*  = "Alerter"
PathName (CIM_STRING/string)  = "I:\WINNT\System32\services.exe"
ProcessId (CIM_UINT32/uint32)  = 228 (0xE4)
ServiceSpecificExitCode (CIM_UINT32/uint32)  = 0 (0x0)
ServiceType (CIM_STRING/string)  = "Share Process"
Started (CIM_BOOLEAN/boolean)  = TRUE
StartMode (CIM_STRING/string)  = "Auto"
StartName (CIM_STRING/string)  = "LocalSystem"
State (CIM_STRING/string)  = "Running"
Status (CIM_STRING/string)  = "OK"
SystemCreationClassName (CIM_STRING/string)  = "Win32_ComputerSystem"
SystemName (CIM_STRING/string)  = "NET-DPEN6400A"
TagId (CIM_UINT32/uint32)  = 0 (0x0)
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)

__CLASS (CIM_STRING/)  = "Win32_Service"
...
__RELPATH (CIM_STRING/)  = "Win32_Service.Name="AppMgmt""
...
__SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
AcceptPause (CIM_BOOLEAN/boolean)  = FALSE
AcceptStop (CIM_BOOLEAN/boolean)  = FALSE
Caption (CIM_STRING/string)  = "Application Management"
CheckPoint (CIM_UINT32/uint32)  = 0 (0x0)
CreationClassName (CIM_STRING/string)  = "Win32_Service"
Description (CIM_STRING/string)  = "Application Management"
DesktopInteract (CIM_BOOLEAN/boolean)  = FALSE
Displayname (CIM_STRING/string)  = "Application Management"
ErrorControl (CIM_STRING/string)  = "Normal"
ExitCode (CIM_UINT32/uint32)  = 1077 (0x435)
InstallDate (CIM_DATETIME/datetime)  = <null>
Name (CIM_STRING/string)*  = "AppMgmt"
PathName (CIM_STRING/string)  = "I:\WINNT\system32\services.exe"
ProcessId (CIM_UINT32/uint32)  = 0 (0x0)
ServiceSpecificExitCode (CIM_UINT32/uint32)  = 0 (0x0)
ServiceType (CIM_STRING/string)  = "Share Process"
Started (CIM_BOOLEAN/boolean)  = FALSE
StartMode (CIM_STRING/string)  = "Manual"
StartName (CIM_STRING/string)  = "LocalSystem"
State (CIM_STRING/string)  = "Stopped"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
__SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
...
Name (CIM_STRING/string)*  = "Browser"
...
```

```
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        __SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
        ...
        Name (CIM_STRING/string)*  = "CertSvc"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        __SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
        ...
        Name (CIM_STRING/string)*  = "cisvc"
        ...
        ...
        ...
```

If you carefully observe the *Name* property, you will see that a star follows the property name. This indicates that this property is a key. As explained in Chapter 2, a key is an instance property used as a unique identifier. This query generates a lot of information. If we are interested only in some of the properties, such as the service *State* (running, stopped, etc.) with the name of all *Win32_Service* instances, the WQL query can be modified as shown in Sample 3.2.

Sample 3.2 *Using the SELECT statement to retrieve some properties of all instances*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Select State, Displayname From Win32_Service"
(WQL) Select State, Displayname From Win32_Service
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Alerter"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Application Management"
        State (CIM_STRING/string)  = "Stopped"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Computer Browser"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Certificate Services"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Indexing Service"
        State (CIM_STRING/string)  = "Stopped"
        ...
        ...
        ...
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Displayname (CIM_STRING/string)  = "Windows Management Instrumentation Driver Extensions"
State (CIM_STRING/string)  = "Running"
```

The list of properties is limited to the two requested properties: *State* and *Displayname*. It is important to note that the **FROM** statement retrieves all instances of the *Win32_Service* class, including all instances derived from the *Win32_Service* class. As the *Win32_Service* class is not derived in the current CIM Schema, we get only the *Win32_Service* instances. Now, for the exercise, you can perform the same query, but instead of using the *Win32_Service* class, you can use the *Win32_BaseService* class. You will see that you will retrieve all *Win32_Service* instances and all *Win32_SystemDriver* instances. The query will look like this:

```
Select * FROM Win32_BaseService
```

To limit the result of the query to one instance type, you can use the system property called *__CLASS* with the **WHERE** statement. In such a case, the query will look like this:

```
Select * FROM Win32_BaseService Where __CLASS="Win32_Service"
```

Due to the filtering effect of the **WHERE** statement, the query will return the exact same list shown in Sample 3.1, because we retrieve only the list of all *Win32_Service* instances available. Instead of retrieving the complete list of the *Win32_Service* instances, we can also reduce the scope to a specific *Win32_Service* instance by using the **WHERE** statement with a different filter. Note that the **WHERE** statement can be used in the three types of query: data, schema, and event. Samples 3.4 and 3.5 looks at the data query case.

Sample 3.3 *Using the WHERE statement to limit the scope to one instance*

```
C:\>wbemdump /Q Root\CIMV2 WQL "Select State, Displayname From Win32_Service Where Name='SNMP'"
(WQL) Select State, Displayname From Win32_Service Where Name='SNMP'
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "SNMP Service"
        State (CIM_STRING/string)  = "Stopped"
```

In data queries the **WHERE** statement refers to a property of the instance that must match the selection criteria. In Sample 3.3 the selection criteria specifies that the *Name* property must match the value "SNMP." Note that the match is case insensitive. We mentioned before that the *Name* property is a key of the *Win32_Service* class. Of course, it is possible to use another property than a key for the selection criteria. For instance, to get the list of all *Win32_Service* instances that are stopped, the query can be formulated as shown in Sample 3.4.

Sample 3.4 *Using the WHERE statement to limit the scope to some instances*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Select Displayname From Win32_Service Where State='Stopped'"
(WQL) Select Displayname From Win32_Service Where State='Stopped'
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Application Management"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Indexing Service"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "ClipBook"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "DHCP Server"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Logical Disk Manager Administrative Service"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Fax Service"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "NetMeeting Remote Desktop Sharing"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Microsoft Exchange Event"

       __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        ...
        ...
        Displayname (CIM_STRING/string)  = "Visual Studio Analyzer RPC bridge"
```

In Sample 3.3, we get only one instance of the *Win32_Service* class. We can reuse this query and extend it to get two instances by using the logical **OR** operator, as shown in Sample 3.5.

Sample 3.5 *Using the WHERE statement to limit the scope to two instances with a logical operator in a WQL data query*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Select State, Displayname From Win32_Service Where Name='SNMP' Or Name='Alerter'"

(WQL) Select State, Displayname From Win32_Service Where Name='SNMP' Or Name='Alerter'
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
```

```
        Displayname (CIM_STRING/string)  = "Alerter"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "SNMP Service"
        State (CIM_STRING/string)  = "Stopped"
```

Our needs for data retrieval may force us to use other logical operators such as **AND** or **NOT**. For example, to retrieve all *Win32_Service* instances other than the Alerter or the SNMP Windows services, we can use the query of Sample 3.6.

Sample 3.6 *Using the NOT logical operator to exclude two instances from a data query*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Select State, Displayname From Win32_Service Where NOT (Name='SNMP' Or Name='Alerter')"

(WQL) Select State, Displayname From Win32_Service Where NOT (Name='SNMP' Or Name='Alerter')
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Application Management"
        State (CIM_STRING/string)  = "Stopped"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Computer Browser"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Certificate Services"
        State (CIM_STRING/string)  = "Running"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "Indexing Service"
        State (CIM_STRING/string)  = "Stopped"

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Displayname (CIM_STRING/string)  = "ClipBook"
        State (CIM_STRING/string)  = "Stopped"
        ...
        ...
        ...
        WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

Notice that the result of the query does not list the Alerter Windows service as made in query Sample 3.2. The SNMP Windows service is not listed either, but, as the output is quite long, we skipped the rest of the display output. If you perform this exercise on your own, you will easily verify this. The query used the **NOT** logical operator for academic purposes. The

same query can be made by changing the matching test and using the **AND** operator:

```
Select State, Displayname From Win32_Service Where (Name<>'SNMP' And Name<>'Alerter')
```

or

```
Select State, Displayname From Win32_Service Where (Name!='SNMP' And Name!='Alerter')
```

Note that two forms of the "not equal to" operator (!= or <>) can be used.

3.2.1.2 Retrieving the associators of a class instance

The second statement that can be used to perform data queries is the **Associators Of** statement. This statement can be used to retrieve the associations related to an instance of a class or to a class only. This means that the **Associators Of** statement can be used for data queries and schema queries. In the previous chapter, to understand the class dependencies, we used the Microsoft Exchange System Attendant (named MSExchangeSA in the system registry). To visualize its associations, we used the **WMI CIM Studio** graphical interface. Now, as we discover WQL, we can also use a WQL query to obtain the exact same information.

Sample 3.7 shows the command line to use with **WBEMDump.exe** and its display output. Note that for readability purposes, the output displays only the *Key* property of the *Win32_Service* class because the WQL query retrieves all the properties of the listed instances.

Sample 3.7 *Retrieving the associated instances of a given instance with the Associators Of WQL statement*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Associators Of {Win32_Service='MSExchangeSA'}"
(WQL) Associators Of {Win32_Service='MSExchangeSA'}
        __CLASS (CIM_STRING/)  = "Win32_ComputerSystem"
        ...
        Name (CIM_STRING/string)*  = "NET-DPEN6400A"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeIS"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeMTA"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
```

```
Name (CIM_STRING/string)*  = "Eventlog"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "NtLmSsp"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

If you look to the service dependencies of the Microsoft Exchange System Attendant visible in Figure 3.1, you will see that the list returned by the WQL query is exactly the same as the one shown with **WMI CIM Studio**. All instances associated with the *Win32_Service* instance MSExchangeSA are listed in the WQL query result. Note that the property used to identify the *Win32_Service* MSExchangeSA instance is the *Name* property defined as a key. To be formal, the query can also be specified as follows:

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Associators Of {Win32_Service.Name='MSExchangeSA'}"
```

This syntax variation can be useful in a case where the instance to be identified has several keys. By explicitly specifying the key name, you select the key to use to perform the test on. For example, the *Win32_Account* class uses two keys: the *Name* and the *Domain* properties. Therefore; the query will be as follows:

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Account.Name='Alain.Lissoir',Domain='LisswareNET'}"
```

For items having only one key, this formal notation is unnecessary as the unique key is used by default to perform the test. Note that this notation is

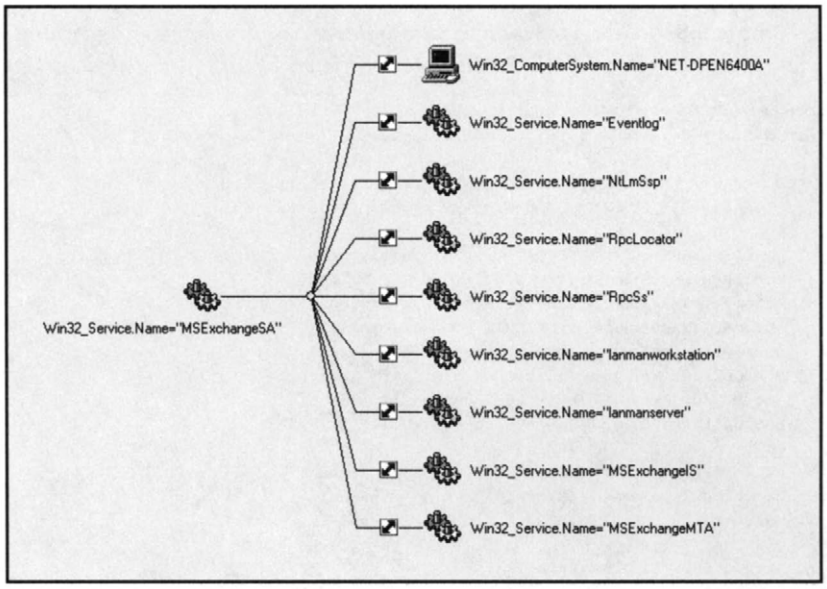

Figure 3.1 *Viewing the Microsoft Exchange 2000 System Attendant service dependencies with WMI CIM Studio.*

valid for data and schema queries. This notation is also applicable to the **References Of** statement that we will see in the next section.

The **Associators Of** statement can be used in combination with the **WHERE** statement. The formal syntax representation of the **Associators Of** statement is as follows:

```
ASSOCIATORS OF {ObjectPath} [WHERE ClassDefsOnly]
                            [WHERE KeysOnly]
                            [WHERE AssocClass = AssocClassName] [ClassDefsOnly]
                            [WHERE RequiredAssocQualifier = QualifierName] [ClassDefsOnly]
                            [WHERE RequiredQualifier = QualifierName] [ClassDefsOnly]
                            [WHERE ResultClass = ClassName] [ClassDefsOnly]
                            [WHERE ResultRole = PropertyName] [ClassDefsOnly]
                            [WHERE Role = PropertyName] [ClassDefsOnly]
```

As we can see, the **Associators Of** statement can be combined with some extra keywords to refine the selection criteria of the WQL query.

The next sample uses the **ClassDefsOnly** keyword and performs the exact same query as Sample 3.7, but the retrieved information is the class definition of the associated instances, instead of the instances themselves. It is important to note that the query is still a data query because it queries an instance (the MSExchangeSA *Win32_Service* instance) and retrieves the classes of the associated instances. Making the same query with the **ClassDefsOnly** keyword will give the result shown in Sample 3.8.

Sample 3.8 *Retrieving the associated instance class definitions of a given instance with the Associators Of WQL statement (Associators Of statement and ClassDefsOnly keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where ClassDefsOnly"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where ClassDefsOnly
        __CLASS (CIM_STRING/)  = "Win32_ComputerSystem"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_UnitaryComputerSystem"
        AdminPasswordStatus (CIM_UINT16/uint16)  = <null>
        AutomaticResetBootOption (CIM_BOOLEAN/boolean)  = <null>
        AutomaticResetCapability (CIM_BOOLEAN/boolean)  = <null>
        BootOptionOnLimit (CIM_UINT16/uint16)  = <null>
        BootOptionOnWatchDog (CIM_UINT16/uint16)  = <null>
        BootROMSupported (CIM_BOOLEAN/boolean)  = <null>
        BootupState (CIM_STRING/string)  = <null>
        Caption (CIM_STRING/string)  = <null>
        ChassisBootupState (CIM_UINT16/uint16)  = <null>
        CreationClassName (CIM_STRING/string)  = <null>
        CurrentTimeZone (CIM_SINT16/sint16)  = <null>
        ...
        Status (CIM_STRING/string)  = <null>
        SupportContactDescription (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        SystemStartupDelay (CIM_UINT16/uint16)  = <null>
        SystemStartupOptions (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        SystemStartupSetting (CIM_UINT8/uint8)  = <null>
        SystemType (CIM_STRING/string)  = <null>
        ThermalState (CIM_UINT16/uint16)  = <null>
        TotalPhysicalMemory (CIM_UINT64/uint64)  = <null>
        UserName (CIM_STRING/string)  = <null>
        WakeUpType (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_Service"        ...
        __SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
        AcceptPause (CIM_BOOLEAN/boolean)  = <null>
        AcceptStop (CIM_BOOLEAN/boolean)  = <null>
        Caption (CIM_STRING/string)  = <null>
        CheckPoint (CIM_UINT32/uint32)  = <null>
        CreationClassName (CIM_STRING/string)  = <null>
        Description (CIM_STRING/string)  = <null>
        ...
        ServiceType (CIM_STRING/string)  = <null>
        Started (CIM_BOOLEAN/boolean)  = <null>
        StartMode (CIM_STRING/string)  = <null>
        StartName (CIM_STRING/string)  = <null>
        State (CIM_STRING/string)  = <null>
        Status (CIM_STRING/string)  = <null>
        SystemCreationClassName (CIM_STRING/string)  = <null>
        SystemName (CIM_STRING/string)  = <null>
        TagId (CIM_UINT32/uint32)  = <null>
        WaitHint (CIM_UINT32/uint32)  = <null>
```

As the query retrieves the class definitions of the associated instances and not the associated instances themselves, there are only two classes listed: the *Win32_ComputerSystem* class and the *Win32_Service* class (see Figure 3.2).

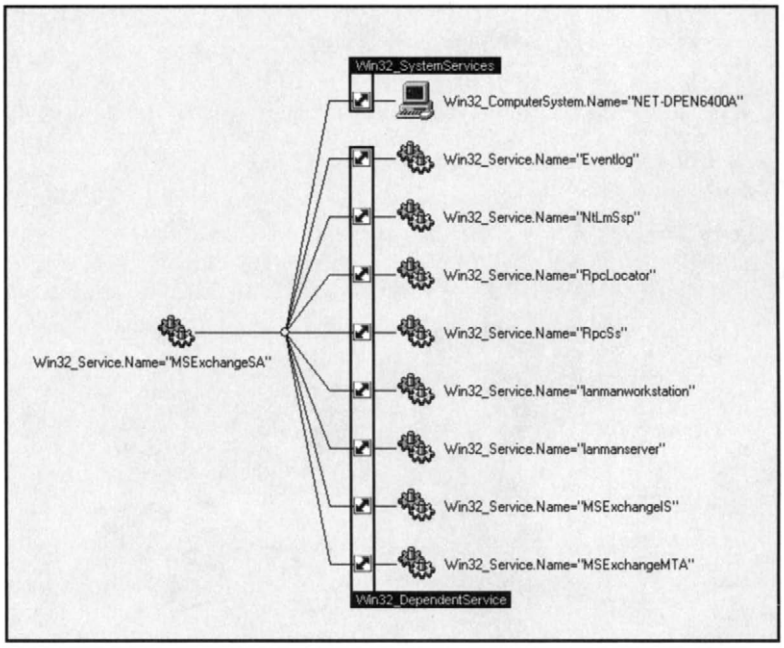

Figure 3.2 *Win32_Service MSExchangeSA instance associated instances and Associations instances.*

If you examine the output of the previous sample (see Sample 3.7), you will see that every instance retrieved uses one of these two classes. These associations are made via association classes called *Win32_SystemServices* and *Win32_DependentService* (see Figure 3.2).

Every time the **ClassDefsOnly** keyword is specified, the query retrieves the class definitions of the instances instead of the instances themselves. This keyword can be combined with the other valid keywords of the **Associators Of** statement.

Now, if we want to retrieve the *Win32_Service* associated instances (which means without the *Win32_ComputerSystem* instance), we can narrow the WQL query to retrieve the associated instance only when the Association instance is made from a *Win32_DependentService* class. This is achieved by using the **AssocClass** keyword, as shown in Sample 3.9.

Sample 3.9 *Retrieving the associated instances of a given instance that are using a particular association class name (Associators Of statement and AssocClass keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where AssocClass=Win32_DependentService"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where AssocClass=Win32_DependentService
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "MSExchangeIS"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "MSExchangeMTA"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "Eventlog"

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "NtLmSsp"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

We see in this query result that only the *Win32_Service* instances are listed because these instances are associated with the association instances made from the *Win32_DependentService* class. The associated instance *Win32_ComputerSystem* (shown in Sample 3.7) is not listed any more with this **Where** filter because this instance is associated with an association instance made from the *Win32_SystemServices* class (see Figure 3.2 for the visual representation of the associations in place).

In this last sample, we filter on the association class name; it is also possible to filter on the presence of a qualifier defined in the association instance. In the previous chapter, we examined the qualifiers of the *Win32_DependentService* association class. This association class had the following qualifiers defined: Association, Description, Dynamic, Locale, Provider, and UUID (see Figure 3.3).

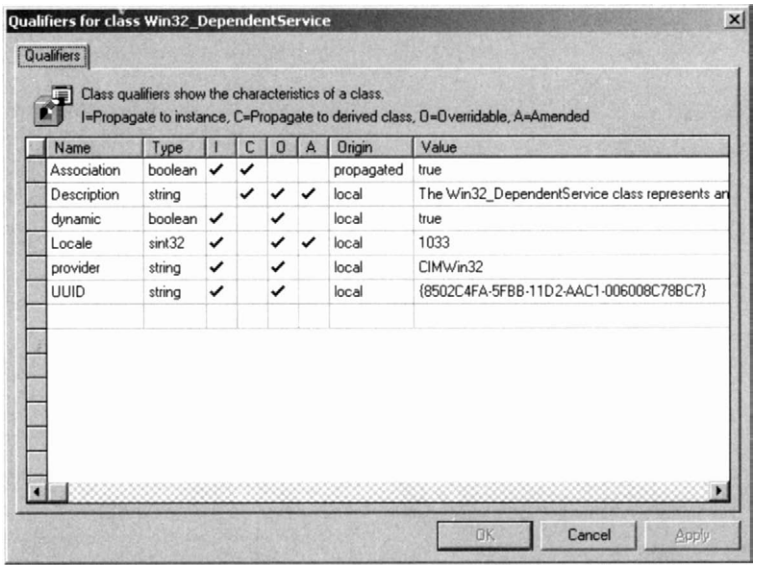

Figure 3.3 *The Win32_DependentService Association class qualifiers.*

Therefore, if we want to get a list of instances using association instances that have a particular qualifier, the WQL query will use the **RequiredAssocQualifier** keyword as shown in Sample 3.10.

Sample 3.10 *Retrieving the associated instances of a given instance where the Association class instances contain a specific qualifier (Associators Of statement and RequiredAssocQualifier keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where RequiredAssocQualifier=Dynamic"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where RequiredAssocQualifier=Dynamic
        __CLASS (CIM_STRING/)  = "Win32_ComputerSystem"
        ...
        Name (CIM_STRING/string)*  = "NET-DPEN6400A"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeIS"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeMTA"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "Eventlog"
        ...
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "NtLmSsp"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
 WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

Of course, in this case, we get the exact same result as Sample 3.7 because both the *Win32_SystemServices* and *Win32_DependentService* Association instances contain the Dynamic qualifier. As both association instances contain the same qualifiers, it is impossible to distinguish them with a query using this qualifier.

This last query filters on the presence of a qualifier in an association instance. So, instead of filtering on a qualifier defined in an association instance, as before, we filter on a qualifier present in the associated instance. The WQL query uses the RequiredQualifier keyword as shown in Sample 3.11.

Sample 3.11 *Retrieving the associated instances of a given instance where the associated instances contain a specific qualifier (Associators Of statement and RequiredQualifier keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where RequiredQualifier=Dynamic"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where RequiredQualifier=Dynamic
        __CLASS (CIM_STRING/)  = "Win32_ComputerSystem"
        ...
        Name (CIM_STRING/string)*  = "NET-DPEN6400A"
        ...
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "MSExchangeIS"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "MSExchangeMTA"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "Eventlog"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "NtLmSsp"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

Again, because the *Win32_ComputerSystem* instance and the *Win32_Service* instances use the Dynamic qualifier, we get the same result as in Sample 3.7. Because the associated instances contain the same qualifiers, it is impossible to distinguish them with a query using this qualifier. However, if one of these two classes were defined as a singleton class (a class that has only one instance in a system), it would be possible to use the **RequiredQualifier** keyword to filter on the presence of the singleton qualifier.

It is possible to obtain the same result as in Sample 3.9 by using the **ResultClass** keyword instead of the **AssocClass** keyword. In this case, the test is performed on the class of the instances associated with the instance given in the WQL query. In Sample 3.9, the test was performed on the class of the association instances. This query with its result is shown in Sample 3.12 (see Figure 3.2 for the visual representation of the used classes).

Sample 3.12 *Retrieving the associated instances of a given instance where the associated instances match a specific class name (Asssociators Of statement and ResultClass keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_Service"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_Service
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeIS"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeMTA"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "Eventlog"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "NtLmSsp"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "RpcLocator"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "RpcSs"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "lanmanworkstation"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "lanmanserver"
        ...
        ...
        ...
        WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

In this sample, we see that the same results can be obtained even if the test uses different information. Instead of testing on the class of the association instances, the test is performed on the class of the Windows service instances.

In regard to the association class *Win32_DependentService*, we saw that associations are made in two ways: the Windows services that depend on the *Win32_Service* instance startup and the Windows services that must be started to start the *Win32_Service* instance. By using the **Associators Of** statement in combination with the **ResultRole** keyword, it is possible to find the list of instances that depend on each other based on the role defined in the associations. For instance, we saw that the MSExchangeMTA and MSExchangeIS services depend on the MSExchangeSA service. By using the **ResultRole** keyword, it is possible to retrieve the service list with a WQL query. The query with its result will be:

Sample 3.13 *Retrieving the associated instances of a given instance based on the associated instances role (Associators Of statement and ResultRole keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where ResultRole=Dependent"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where ResultRole=Dependent
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeIS"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeMTA"
        ...
        ...
        ...
        WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

Therefore, if we want to retrieve the *Win32_Service* instances that must be started in order to start the MSExchangeSA *Win32_Service* instance, we must use the following query:

Sample 3.14 *Retrieving the associated instances of a given instance based on the associated instances role (Associators Of statement and ResultRole keyword) (complementary query)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where ResultRole=Antecedent"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where ResultRole=Antecedent
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "Eventlog"
        ...
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "NtLmSsp"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

In the last two WQL queries (Sample 3.13 and 3.14), filtering is performed on the role played by the associated instances in relation to the given *Win32_Service* instance. In the same way, it is possible to filter the

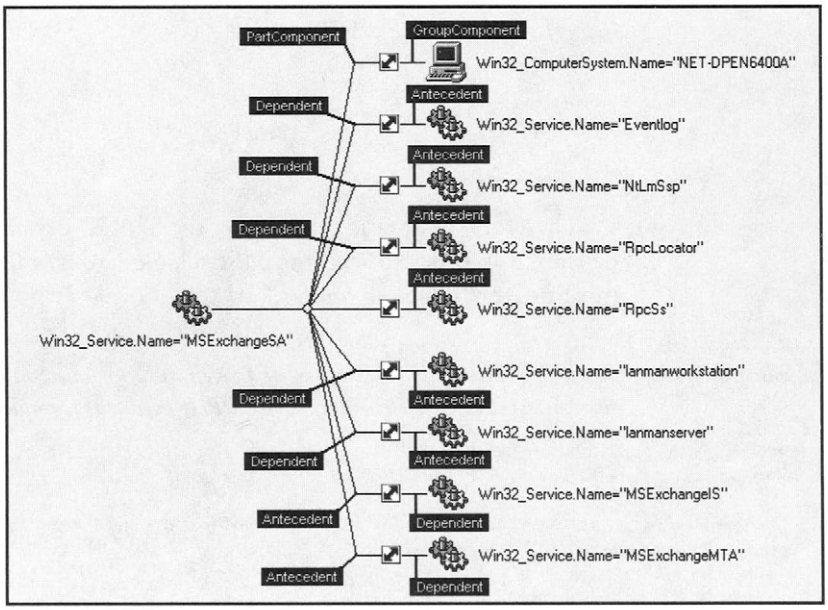

Figure 3.4
The role of the association between the given instance and the associated instances.

associated instances based on the role played by the given *Win32_Service* instance. This query is the complementary result of the two previous queries, as the MSExchangeSA is an antecedent Windows service for the MSExchangeIS and MSExchangeMTA Windows services, and MSExchangeIS and MSExchangeMTA are dependent Windows services of the MSExchangeSA Windows service (see Figure 3.4).

As a reminder, Section 2.9.1 explains what antecent and dependent Windows services are. To obtain the result of Sample 3.13 with the role played by the given instance, we must use the **Role** keyword as shown in Sample 3.15.

Sample 3.15 *Retrieving the associated instances of a given instance based on the given instance role (Associators Of statement and Role keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where Role=Antecedent"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where Role=Antecedent
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeIS"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "MSExchangeMTA"
        ...
        ...
        ...
        WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

The query shown in Sample 3.16 produces a complementary result for Sample 3.15 (same result as Sample 3.14).

Sample 3.16 *Retrieving the associated instances of a given instance based on the given instance role (Associators Of statement and Role keyword) (complementary query)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service='MSExchangeSA'} Where Role=Dependent"

(WQL) Associators Of {Win32_Service='MSExchangeSA'} Where Role=Dependent
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "Eventlog"
        ...

        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        Name (CIM_STRING/string)*  = "NtLmSsp"
        ...
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcLocator"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "RpcSs"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanworkstation"
...

__CLASS (CIM_STRING/)  = "Win32_Service"
...
Name (CIM_STRING/string)*  = "lanmanserver"
...
...
...
WaitHint (CIM_UINT32/uint32)  = 0 (0x0)
```

It is important to note that an **Associators Of** statement that uses the **WHERE** statement cannot use the **AND** or the **OR** logical operator to separate keywords. However, any of the keywords can be combined in the same query to have an implicit **AND** logical operation. This important detail shows that it is crucial not to confuse WQL with SQL even if there are some similarities. For instance, you may have the following:

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
   "Associators Of {Win32_Service} Where ResultClass=Win32_ServiceSpecification
                                         RequiredQualifier=Dynamic SchemaOnly"
```

Furthermore, the equal sign is the only valid operator that can be used with the keywords in these queries.

As we have seen in this section, the last 12 samples deal with associations. When we talk about associations, this always implies the use of references. It is possible to perform data queries that take advantage of the defined references in the associations, which leads us to the next section.

3.2.1.3 Retrieving the references of a class instance

To define the association between a given *Win32_Service* instance and any other *Win32_Service* instances, two specific properties of the *Win32_DependentService* Association class are used: the *Dependent* property and the *Antecedent* property. As we saw in Section 2.9.1, these properties are called the references. It is possible to retrieve the references of an association instance by browsing with the **WMI CIM Studio**, but it is also possible to retrieve the references of an association instance with a WQL query. The

statement used to perform this data query is the **References Of** statement. This is the third statement type able to perform a data query. Note that as with the **Associators Of** statement, the **References Of** statement can also be used for WQL schema queries. We revisit schema queries in Section 3.2.2.

In the previous sections, we retrieved the instances associated with the given *Win32_Service* instance. Even if the information retrieved is related to the associated instances, behind the scenes, the association instances play an important role. Here, we retrieve the association instances with their related properties (which are the references). Sample 3.17 performs this data query using the **References Of** statement.

Sample 3.17 *Retrieving the Association class instances of a given instance with the References Of WQL statement*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "References Of {Win32_Service='MSExchangeSA'}"
(WQL) References Of {Win32_Service='MSExchangeSA'}
        __CLASS (CIM_STRING/)  = "Win32_SystemServices"
        ...
        GroupComponent (CIM_REFERENCE/ref:Win32_ComputerSystem)*  =
                        "\\MYSERVER\root\cimv2:Win32_ComputerSystem.Name="NET-DPEN6400A""
        PartComponent (CIM_REFERENCE/ref:Win32_Service)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeIS""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeMTA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="Eventlog""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="NtLmSsp""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                        "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

```
__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
  Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="RpcLocator""
       Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="RpcSs""
__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="lanmanworkstation""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="lanmanserver""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

As we can see in the query result, we retrieve all association instances related to the given *Win32_Service* instance. In the property list of association instances, we find the references (see Figure 3.4):

- *GroupComponent* and *PartComponent* for the *Win32_SystemServices* Association instance
- *Antecedent* and *Dependent* for the *Win32_DependentService* association instance

These references point to the associated instances. Each reference contains the full WMI path of the associated instance. For any associated *Win32_Service* instances, based on the dependency type, the associated instances are assigned to the *Antecedent* or *Dependent* reference. Notice that since the references are the unique identifiers of the association instances, they are marked with a star to indicate that these properties are defined as Keys.

The formal syntax of the **References Of** statement is as follows:

```
REFERENCES OF {SourceObject} [WHERE ClassDefsOnly]
                             [WHERE KeysOnly]
                             [WHERE RequiredQualifier = QualifierName] [ClassDefsOnly]
                             [WHERE ResultClass = ClassName] [ClassDefsOnly]
                             [WHERE Role = PropertyName] [ClassDefsOnly]
```

The **References Of** statement is similar to the **Associators Of** statement since it can also be combined with some extra keywords to refine the selection criteria of the WQL query. Although we retrieve some of the keywords already seen for the **Associators Of** statement, the **References Of** statement retrieves the properties of the association instances.

When executing the query shown in Sample 3.17, we retrieve the association instances. With the **ClassDefsOnly** keyword, the query retrieves the association class definitions of these association instances instead of the association instances themselves. An example of this type of query is shown in Sample 3.18.

Sample 3.18 *Retrieving the Association instance class definitions of a given instance with the References Of WQL statement (References Of statement and ClassDefsOnly keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "References Of {Win32_Service='MSExchangeSA'} Where ClassDefsOnly"
WQL) References Of {Win32_Service='MSExchangeSA'} Where ClassDefsOnly
        __CLASS (CIM_STRING/)  = "Win32_SystemServices"
        ...         __SUPERCLASS (CIM_STRING/)  = "CIM_SystemComponent"
        GroupComponent (CIM_REFERENCE/ref:Win32_ComputerSystem)*  = <null>
        PartComponent (CIM_REFERENCE/ref:Win32_Service)*  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  = <null>
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  = <null>
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

If you examine the output from Sample 3.17 you will notice that every association instance uses one of the association class definitions listed in the output result of Sample 3.18. The principle of using the **ClassDefsOnly** keyword with the **References Of** statement is exactly the same as we saw in the context of the **Associators Of** statement. Every time the **ClassDefsOnly** keyword is specified, the query retrieves the class definitions of the instances (instead of the instances themselves). This keyword can be combined with the other valid keywords of the **References Of** statement.

The **RequiredQualifier** keyword (used in Sample 3.19) has the exact same role as the one we saw for the **Associators Of** statement. The **RequiredQualifier** keyword verifies the presence of a specific qualifier in the examined instances. Here, however, it looks for the presence of the qualifier in the association instances and not in the associated instances. So the **RequiredQualifier** keyword is the same keyword as the one used previously (see Sample 3.11), but because we use the **References Of** statement (instead of the **Associators Of** statement), the examined instance is different. To summarize: when the **RequiredQualifier** keyword is used with the **Associa-**

tors Of statement it looks for a qualifier in the associated instances; when the **RequiredQualifier** keyword is used with the **References Of** statement it looks for a qualifier in the association instances. The **References Of** statement retrieves the association instances, and the **Associators Of** statement retrieves the associated instances. Let's take a look at Sample 3.19 below (see Figure 3.2 for the visual representation of the associations in place).

Sample 3.19 *Retrieving the Association class instances of a given instance when the Association class instances contain a particular qualifier (References Of statement and RequiredQualifier keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "References Of {Win32_Service='MSExchangeSA'} Where RequiredQualifier=Dynamic"
(WQL) References Of {Win32_Service='MSExchangeSA'} Where RequiredQualifier=Dynamic
        __CLASS (CIM_STRING/)  = "Win32_SystemServices"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_SystemComponent"
         GroupComponent (CIM_REFERENCE/ref:Win32_ComputerSystem)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_ComputerSystem.Name="W2K-DPEN6400""
        PartComponent (CIM_REFERENCE/ref:Win32_Service)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =


                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeIS""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeMTA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="Eventlog""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="NtLmSsp""
```

```
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
       "TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcLocator""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
       "TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcSs""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanworkstation""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanserver""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

Again, because both association instances of a *Win32_Service* instance contain the same qualifiers, the complete list of association instances related to the given *Win32_Service* instance is returned by the query. This means that we get the exact same output as Sample 3.17, which retrieved all the association instances. For example, if the *Win32_DependentService* association class had a different qualifier than the *Win32_SystemServices* association class, it would be possible to match a different qualifier name with the WQL query to retrieve results based on this criteria. But as these association instances use the same qualifiers, we retrieve all association instances.

When using the **References Of** statement in combination with the **ResultClass** keyword, it is possible to select the association instances to be retrieved by the WQL query based on the association class used by the asso-

ciation instances (see Sample 3.20). Again, see Figure 3.2 for the visualrepresentation of the associations in place.

Sample 3.20 *Retrieving the Association class instances of a given instance where the Association class instances match a given class name (References Of statement and ResultClass keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "References Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_SystemServices"
(WQL) References Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_SystemServices
        __CLASS (CIM_STRING/)  = "Win32_SystemServices"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_SystemComponent"
        GroupComponent (CIM_REFERENCE/ref:Win32_ComputerSystem)*  =
             "\\MYSERVER\root\cimv2:Win32_ComputerSystem.Name="NET-DPEN6400A""
        PartComponent (CIM_REFERENCE/ref:Win32_Service)*  =
             "\\MYSERVER\root\cimv2:Win32_Service.Name="MSExchangeSA""
```

Sample 3.21 uses the exact same query as Sample 3.20, but it uses the other association class name to retrieve the complementary result of the previous query.

Sample 3.21 *Retrieving the Association class instances of a given instance where the Association class instances match a given class name (References Of statement and ResultClass keyword) (complementary query)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "References Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_DependentService"
(WQL) References Of {Win32_Service='MSExchangeSA'} Where ResultClass=Win32_DependentService
        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeIS""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeMTA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="Eventlog""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

```
__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="NtLmSsp""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>
__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcLocator""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcSs""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanworkstation""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null>

__CLASS (CIM_STRING/)  = "Win32_DependentService"
...
__SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanserver""
Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
          "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
TypeOfDependency (CIM_UINT16/uint16)  = <null
```

It is also possible to retrieve the association instances based on the role played by the given instance in the association. For this, the **Role** keyword must be used as shown in Sample 3.22.

Sample 3.22 *Retrieving the Association class instances of a given instance where the given instance plays a specific role name in the Association instance (References Of statement and Role keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "References Of {Win32_Service='MSExchangeSA'} Where Role=Antecedent"

(WQL) References Of {Win32_Service='MSExchangeSA'} Where Role=Antecedent
        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
```

```
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeIS""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeMTA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

In Sample 3.22 we have two references in the association that are based on the role played by the *Win32_Service* instance. Sample 3.23 retrieves the complementary query result of Sample 3.22. Once more, see Figure 3.4 for the visual representation of the roles in place.

Sample 3.23 *Retrieving the Association class instances of a given instance where the given instance plays a specific role name in the Association instance (References Of statement and Role keyword) (complementary query)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "References Of {Win32_Service='MSExchangeSA'} Where Role=Dependent"

(WQL) References Of {Win32_Service='MSExchangeSA'} Where Role=Dependent
        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="Eventlog""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="NtLmSsp""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcLocator""
```

```
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="RpcSs""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
"\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanworkstation""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_DependentService"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_ServiceServiceDependency"
        Antecedent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="lanmanserver""
        Dependent (CIM_REFERENCE/ref:Win32_BaseService)*  =
                  "\\W2K-DPEN6400\root\cimv2:Win32_Service.Name="MSExchangeSA""
        TypeOfDependency (CIM_UINT16/uint16)  = <null>
```

It is important to note that the **WHERE** statement that uses the **References Of** statement cannot use the **AND** or the **OR** logical operators to separate keywords. But, similar to the **Associators Of** statement, keywords can be combined in the same query. Furthermore, the equal sign is the only valid operator that can be used with the keywords in these queries.

Although quite abstract, this notion of associations and references between instances makes more sense when performing WQL queries. Since these data queries can be quite confusing at first glance, Table 3.3 summarizes the different WQL data-query types with the obtained results.

3.2.2 Schema queries

With the previous category of query, we focused on the real-world manageable entities themselves. Manageable entities are nothing more than instances of classes; therefore it can be interesting to perform queries on the classes themselves. Classes are the templates of real-world manageable enti-

Table 3.3 *WQL Data Queries*

Statements	Keywords				Obtained Results
Select * | *PropertyName*	From	{*SourceClass*}	–	–	All instances of the *SourceClass* and instances derived from the *SourceClass*.
	From	{*SourceClass*}	Where	*PropertyName* = *Value*	Any instances of the *SourceClass* matching the selection criteria.
Associators Of	–	{*SourceObject*}	–	–	Any instances associated with the *SourceObject*.
	–	{*SourceObject*}	Where	ClassDefsOnly	Forces to retrieve any instance class definitions (instead of the instances themselves) that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	AssocClass = *AssocClassName*	Any instances where the Association instances have a class name matching *AssocClassName* and that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	RequiredAssocQualifier = *QualifierName*	Any instances where the Association instances are using the *QualifierName* and that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	RequiredQualifier = *QualifierName*	Any instances containing the *QualifierName* and that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	ResultClass = *ClassName*	Any instances having a class name matching the *ClassName* and that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	ResultRole = *PropertyName*	Any instances where the role of the instances in the association matches the *PropertyName* and that are associated with the *SourceObject*.
	–	{*SourceObject*}	Where	Role = *PropertyName*	Any instances where the role of the *SourceObject* matches the *PropertyName* and that are associated with the *SourceObject*.

Table 3.3 *WQL Data Queries (continued)*

Statements	Keywords				Obtained Results
References Of	–	{*SourceObject*}	–	–	Any Association instances doing an Association with the *SourceObject*.
	–	{*SourceObject*}	Where	ClassDefsOnly	Any Association instance class definitions doing an Association with the *SourceObject*.
	–	{*SourceObject*}	Where	RequiredQualifier = *QualifierName*	Any Association instances containing the *QualifierName* and been an Association instance for the *SourceObject*.
	–	{*SourceObject*}	Where	ResultClass = *ClassName*	Any Association instances matching the *ClassName* and been an Association instance for the *SourceObject*.
	–	{*SourceObject*}	Where	Role = *PropertyName*	Any Association instances where the *SourceObject* plays a role matching the *PropertyName*.

ties. When we examine the class definitions, we are actually examining the CIM repository schema, which leads us to the topic of schema queries.

3.2.2.1 Retrieving a class

In the previous section we retrieved instances of the *Win32_Service* class by using the following query:

```
Select * From Win32_Service
```

With such a WQL query, the retrieved information always relates to instances. The data queries examined so far have not retrieved the schema definition of the queried instance. In order to retrieve the schema definition of the queried instance, we need to use a schema query. Schema queries retrieve the CIM schema definition of a class. For example, to retrieve the schema definition of the *Win32_Service* class with a schema query, we must use a WQL query as shown in Sample 3.24.

Sample 3.24 *Retrieving the class definition of a given class with the SELECT statement*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "SELECT * FROM meta_class WHERE __this ISA 'Win32_Service'"
(WQL) SELECT * FROM meta_class WHERE __this ISA 'Win32_Service'
        __CLASS (CIM_STRING/)  = "Win32_Service"
        ...
        __SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
        AcceptPause (CIM_BOOLEAN/boolean)  = <null>
        AcceptStop (CIM_BOOLEAN/boolean)  = <null>
        Caption (CIM_STRING/string)  = <null>
        CheckPoint (CIM_UINT32/uint32)  = <null>
        CreationClassName (CIM_STRING/string)  = <null>
        Description (CIM_STRING/string)  = <null>
        DesktopInteract (CIM_BOOLEAN/boolean)  = <null>
        Displayname (CIM_STRING/string)  = <null>
        ErrorControl (CIM_STRING/string)  = <null>
        ExitCode (CIM_UINT32/uint32)  = <null>
        InstallDate (CIM_DATETIME/datetime)  = <null>
        Name (CIM_STRING/string)*  = <null>
        PathName (CIM_STRING/string)  = <null>
        ProcessId (CIM_UINT32/uint32)  = <null>
        ServiceSpecificExitCode (CIM_UINT32/uint32)  = <null>
        ServiceType (CIM_STRING/string)  = <null>
        Started (CIM_BOOLEAN/boolean)  = <null>
        StartMode (CIM_STRING/string)  = <null>
        StartName (CIM_STRING/string)  = <null>
        State (CIM_STRING/string)  = <null>
        Status (CIM_STRING/string)  = <null>
        SystemCreationClassName (CIM_STRING/string)  = <null>
        SystemName (CIM_STRING/string)  = <null>
        TagId (CIM_UINT32/uint32)  = <null>
         WaitHint (CIM_UINT32/uint32)  = <null>
```

The result of the query produces a list of classes made from the *Win32_Service* class specified. With the "*," all the properties of the class definition are listed. In a schema query, when using the **SELECT** statement, there is no other option than using the "*" to retrieve the properties. A schema query does not support the selection of some properties as a data query.

In this query, we have three new keywords: the **ISA** keyword, the **meta_class** keyword, and the **__THIS** keyword. Let's talk about the **ISA** keyword first. In the context of a schema query, the **ISA** keyword retrieves all the class definitions using the class type given in the query and all the subclasses derived from the class type given. In the context of Sample 3.24, since the *Win32_Service* has no subclass, this is the only class retrieved.

Another new keyword is the **meta_class** keyword. This keyword performs the magic of retrieving the class definitions instead of the class instance data. If you execute the previous query without specifying a **WHERE** statement, the query will retrieve the class definition of all classes present in the given namespace. Once again, the **WHERE** statement narrows the scope of the query. Removing the **WHERE** statement and performing the query with the **meta_class** statement enlarges the scope of the classes retrieved. The query will resemble the following:

```
SELECT * FROM meta_class
```

The produced output is too large to show in this book, but we recommend you perform this query to see the classes available in the CIM repository (in a given namespace like **Root\CIMv2** as used in previous samples). You will retrieve all the classes you are able to browse when using the **WMI CIM Studio**.

The last new keyword we need to talk about is the **__THIS** keyword. This keyword is used only in schema queries and identifies the target class for a schema query. In the case of the previous samples, the **__THIS** keyword identifies a target class that must be a *Win32_Service* class (see Sample 3.24).

Previously, we saw that the *Win32_Service* is a subclass of the *Win32_BaseService* class. We also know that the *Win32_BaseService* has two subclasses: *Win32_Service* and *Win32_SystemDrive*. If we repeat the same query but instead of using the *Win32_Service* class we use the *Win32_BaseService* class, we receive the results shown in Sample 3.25.

Sample 3.25 *Retrieving the class definition with the subclasses of a given class with the SELECT statement*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "SELECT * FROM meta_class WHERE __this ISA 'Win32_BaseService'"
(WQL) SELECT * FROM meta_class WHERE __this ISA 'Win32_BaseService'
        __CLASS (CIM_STRING/) = "Win32_BaseService"
        ...
        __SUPERCLASS (CIM_STRING/) = "CIM_Service"
        AcceptPause (CIM_BOOLEAN/boolean) = <null>
        AcceptStop (CIM_BOOLEAN/boolean) = <null>
        Caption (CIM_STRING/string) = <null>
        CreationClassName (CIM_STRING/string) = <null>
        Description (CIM_STRING/string) = <null>
        DesktopInteract (CIM_BOOLEAN/boolean) = <null>
        Displayname (CIM_STRING/string) = <null>
        ErrorControl (CIM_STRING/string) = <null>
        ExitCode (CIM_UINT32/uint32) = <null>
        InstallDate (CIM_DATETIME/datetime) = <null>
        Name (CIM_STRING/string)* = <null>
        PathName (CIM_STRING/string) = <null>
        ServiceSpecificExitCode (CIM_UINT32/uint32) = <null>
        ServiceType (CIM_STRING/string) = <null>
        Started (CIM_BOOLEAN/boolean) = <null>
        StartMode (CIM_STRING/string) = <null>
        StartName (CIM_STRING/string) = <null>
        State (CIM_STRING/string) = <null>
        Status (CIM_STRING/string) = <null>
        SystemCreationClassName (CIM_STRING/string) = <null>
        SystemName (CIM_STRING/string) = <null>
        TagId (CIM_UINT32/uint32) = <null>

         __CLASS (CIM_STRING/) = "Win32_SystemDriver"
        ...
        __SUPERCLASS (CIM_STRING/) = "Win32_BaseService"
        AcceptPause (CIM_BOOLEAN/boolean) = <null>
        AcceptStop (CIM_BOOLEAN/boolean) = <null>
        Caption (CIM_STRING/string) = <null>
        CreationClassName (CIM_STRING/string) = <null>
        Description (CIM_STRING/string) = <null>
        DesktopInteract (CIM_BOOLEAN/boolean) = <null>
        Displayname (CIM_STRING/string) = <null>
        ErrorControl (CIM_STRING/string) = <null>
        ExitCode (CIM_UINT32/uint32) = <null>
        InstallDate (CIM_DATETIME/datetime) = <null>
        Name (CIM_STRING/string)* = <null>
        PathName (CIM_STRING/string) = <null>
        ServiceSpecificExitCode (CIM_UINT32/uint32) = <null>
        ServiceType (CIM_STRING/string) = <null>
        Started (CIM_BOOLEAN/boolean) = <null>
        StartMode (CIM_STRING/string) = <null>
        StartName (CIM_STRING/string) = <null>
        State (CIM_STRING/string) = <null>
        Status (CIM_STRING/string) = <null>
        SystemCreationClassName (CIM_STRING/string) = <null>
        SystemName (CIM_STRING/string) = <null>
        TagId (CIM_UINT32/uint32) = <null>
```

```
__CLASS (CIM_STRING/)  = "Win32_Service"
...
__SUPERCLASS (CIM_STRING/)  = "Win32_BaseService"
AcceptPause (CIM_BOOLEAN/boolean)  = <null>
AcceptStop (CIM_BOOLEAN/boolean)  = <null>
Caption (CIM_STRING/string)  = <null>
CheckPoint (CIM_UINT32/uint32)  = <null>
CreationClassName (CIM_STRING/string)  = <null>
Description (CIM_STRING/string)  = <null>
DesktopInteract (CIM_BOOLEAN/boolean)  = <null>
Displayname (CIM_STRING/string)  = <null>
ErrorControl (CIM_STRING/string)  = <null>
ExitCode (CIM_UINT32/uint32)  = <null>
InstallDate (CIM_DATETIME/datetime)  = <null>
Name (CIM_STRING/string)*  = <null>
PathName (CIM_STRING/string)  = <null>
ProcessId (CIM_UINT32/uint32)  = <null>
ServiceSpecificExitCode (CIM_UINT32/uint32)  = <null>
ServiceType (CIM_STRING/string)  = <null>
Started (CIM_BOOLEAN/boolean)  = <null>
StartMode (CIM_STRING/string)  = <null>
StartName (CIM_STRING/string)  = <null>
State (CIM_STRING/string)  = <null>
Status (CIM_STRING/string)  = <null>
SystemCreationClassName (CIM_STRING/string)  = <null>
SystemName (CIM_STRING/string)  = <null>
TagId (CIM_UINT32/uint32)  = <null>
 WaitHint (CIM_UINT32/uint32)  = <null>
```

In this case, the **ISA** keyword allows us to retrieve the class definition of the *Win32_BaseService* class with its two subclasses: *Win32_Service* and *Win32_SystemDriver.*

3.2.2.2 Retrieving the associators of a class

Schema associations determine how instance associations behave. The **Associators Of** and **References Of** statements are applicable to schema queries; however, instead of returning associated instances or association instances, the schema query returns associated class definitions (with the **Associators Of** statement) or association class definitions (with the **References Of** statement). In this context, there are two differences in the schema queries when compared with data queries:

- The source object given in a data query (which is an instance) must be a class rather than an instance.
- The **ClassDefsOnly** keyword is no longer valid and the keyword **SchemaOnly** must be used instead.

The formal notation for the **Associators Of** statement is as follows:

```
ASSOCIATORS OF {SourceClass} [WHERE SchemaOnly]
                             [WHERE AssocClass = AssocClassName] [SchemaOnly]
                             [WHERE RequiredAssocQualifier = QualifierName] [SchemaOnly]
```

```
[WHERE RequiredQualifier = QualifierName] [SchemaOnly]
[WHERE ResultClass = ClassName] [SchemaOnly]
[WHERE ResultRole = PropertyName] [SchemaOnly]
[WHERE Role = PropertyName] [SchemaOnly]
```

The formal notation for the **References Of** statement is as follows:

```
REFERENCES OF {SourceClass} [WHERE SchemaOnly]
                            [WHERE ResultClass = ClassName] [SchemaOnly]
                            [WHERE Role = PropertyName] [SchemaOnly]
                            [WHERE RequiredQualifier = QualifierName] [SchemaOnly]
```

Sample 3.26 *Retrieving the associated class definitions with the Associators Of statement (Associators Of statement and SchemaOnly keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL "Associators Of {Win32_Service} Where SchemaOnly"
(WQL) Associators Of {Win32_Service} Where SchemaOnly
        __CLASS (CIM_STRING/)  = "Win32_ComputerSystem"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_UnitaryComputerSystem"
        AdminPasswordStatus (CIM_UINT16/uint16)  = <null>
        AutomaticResetBootOption (CIM_BOOLEAN/boolean)  = <null>
        AutomaticResetCapability (CIM_BOOLEAN/boolean)  = <null>
        BootOptionOnLimit (CIM_UINT16/uint16)  = <null>
        BootOptionOnWatchDog (CIM_UINT16/uint16)  = <null>
        BootROMSupported (CIM_BOOLEAN/boolean)  = <null>
        BootupState (CIM_STRING/string)  = <null>
        Caption (CIM_STRING/string)  = <null>
        ChassisBootupState (CIM_UINT16/uint16)  = <null>
        CreationClassName (CIM_STRING/string)  = <null>
        CurrentTimeZone (CIM_SINT16/sint16)  = <null>
        DaylightInEffect (CIM_BOOLEAN/boolean)  = <null>
        Description (CIM_STRING/string)  = <null>
        ...
        PrimaryOwnerName (CIM_STRING/string)  = <null>
        ResetCapability (CIM_UINT16/uint16)  = <null>
        ResetCount (CIM_SINT16/sint16)  = <null>
        ResetLimit (CIM_SINT16/sint16)  = <null>
        Roles (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        Status (CIM_STRING/string)  = <null>
        SupportContactDescription (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        SystemStartupDelay (CIM_UINT16/uint16)  = <null>
        SystemStartupOptions (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        SystemStartupSetting (CIM_UINT8/uint8)  = <null>
        SystemType (CIM_STRING/string)  = <null>
        ThermalState (CIM_UINT16/uint16)  = <null>
        TotalPhysicalMemory (CIM_UINT64/uint64)  = <null>
        UserName (CIM_STRING/string)  = <null>
        WakeUpType (CIM_UINT16/uint16)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_WMISetting"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_Setting"
        ASPScriptDefaultNamespace (CIM_STRING/string)  = \\root\cimv2
        ASPScriptEnabled (CIM_BOOLEAN/boolean)  = <null>
        AutorecoverMofs (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        AutoStartWin9X (CIM_UINT32/uint32)  = <null>
        BackupInterval (CIM_UINT32/uint32)  = <null>
```

```
        BackupLastTime (CIM_DATETIME/datetime)  = <null>
              BuildVersion (CIM_STRING/string)  = <null>
       ...
        MaxLogFileSize (CIM_UINT32/uint32)  = <null>
        MaxWaitOnClientObjects (CIM_UINT32/uint32)  = <null>
        MaxWaitOnEvents (CIM_UINT32/uint32)  = <null>
        MofSelfInstallDirectory (CIM_STRING/string)  = <null>
        SettingID (CIM_STRING/string)  = <null>

        __CLASS (CIM_STRING/)  = "Win32_ServiceSpecification"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_Check"
        Caption (CIM_STRING/string)  = <null>
        CheckID (CIM_STRING/string)*  = <null>
        CheckMode (CIM_BOOLEAN/boolean)  = <null>
        Dependencies (CIM_STRING/string)  = <null>
        Description (CIM_STRING/string)  = <null>
        Displayname (CIM_STRING/string)  = <null>
        ErrorControl (CIM_SINT32/sint32)  = <null>
        ID (CIM_STRING/string)  = <null>
        LoadOrderGroup (CIM_STRING/string)  = <null>
        Name (CIM_STRING/string)  = <null>
        Password (CIM_STRING/string)  = <null>
        ServiceType (CIM_SINT32/sint32)  = <null>
        SoftwareElementID (CIM_STRING/string)  = <null>
        SoftwareElementState (CIM_UINT16/uint16)  = <null>
        StartName (CIM_STRING/string)  = <null>
        StartType (CIM_SINT32/sint32)  = <null>
        TargetOperatingSystem (CIM_UINT16/uint16)  = <null>
         Version (CIM_STRING/string)  = <null>
```

In Sample 3.26 output, we see that the class definitions retrieved are directly associated to the *Win32_Service* class: *Win32_ComputerSystem*, *Win32_WMISetting*, and *Win32_ServiceSpecification*. Now, if you want to retrieve only the classes that are associated with the *Win32_Service* class with a particular association class, you must use the **AssocClass** keyword. The query and its results are shown in Sample 3.27.

Sample 3.27 *Retrieving the associated class definitions with the Associators Of statement (Associators Of statement and AssocClass keyword)*

```
C:\>wbemdump /E /Q Root\CIMV2 WQL
    "Associators Of {Win32_Service} Where AssocClass=Win32_WMIElementSetting SchemaOnly"
(WQL) Associators Of {Win32_Service} Where AssocClass=Win32_WMIElementSetting SchemaOnly
        __CLASS (CIM_STRING/)  = "Win32_WMISetting"
        ...
        __SUPERCLASS (CIM_STRING/)  = "CIM_Setting"
        ASPScriptDefaultNamespace (CIM_STRING/string)  = \\root\cimv2
        ASPScriptEnabled (CIM_BOOLEAN/boolean)  = <null>
        AutorecoverMofs (CIM_STRING | CIM_FLAG_ARRAY/string)  = <null>
        AutoStartWin9X (CIM_UINT32/uint32)  = <null>
        BackupInterval (CIM_UINT32/uint32)  = <null>
        BackupLastTime (CIM_DATETIME/datetime)  = <null>
        BuildVersion (CIM_STRING/string)  = <null>
        Caption (CIM_STRING/string)  = <null>
        DatabaseDirectory (CIM_STRING/string)  = <null>
```

```
        DatabaseMaxSize (CIM_UINT32/uint32)  = <null>
        Description (CIM_STRING/string)  = <null>
        EnableAnonWin9xConnections (CIM_BOOLEAN/boolean)  = <null>
        EnableEvents (CIM_BOOLEAN/boolean)  = <null>
        EnableStartupHeapPreallocation (CIM_BOOLEAN/boolean)  = <null>
        HighThresholdOnClientObjects (CIM_UINT32/uint32)  = <null>
        HighThresholdOnEvents (CIM_UINT32/uint32)  = <null>
        InstallationDirectory (CIM_STRING/string)  = <null>
        LastStartupHeapPreallocation (CIM_UINT32/uint32)  = <null>
        LoggingDirectory (CIM_STRING/string)  = <null>
        LoggingLevel (CIM_UINT32/uint32)  = <null>
        LowThresholdOnClientObjects (CIM_UINT32/uint32)  = <null>
        LowThresholdOnEvents (CIM_UINT32/uint32)  = <null>
        MaxLogFileSize (CIM_UINT32/uint32)  = <null>
        MaxWaitOnClientObjects (CIM_UINT32/uint32)  = <null>
        MaxWaitOnEvents (CIM_UINT32/uint32)  = <null>
        MofSelfInstallDirectory (CIM_STRING/string)  = <null>
        SettingID (CIM_STRING/string)  = <null>
```

Note the presence of the **SchemaOnly** keyword. In the context of a schema query that uses the **Associators Of** or the **References Of** statements, the **SchemaOnly** keyword must be specified, otherwise, nothing is returned from the query.

For other keywords that can be used with the **Associators Of** statement, the logic is the same as that for the data queries; therefore, the same rules apply. You can then refer to the data queries section to see the use of these keywords.

3.2.2.3 *Retrieving the references of a class*

By using the **References Of** statement in the context of a schema query, we retrieve the association class definitions.

Sample 3.28 *Retrieving the Association class definitions with the References Of statement (References Of statement and SchemaOnly keyword)*

```
I:\>wbemdump /E /Q Root\CIMV2 WQL "References Of {Win32_Service} Where SchemaOnly"
(WQL) References Of {Win32_Service} Where SchemaOnly
        __CLASS (CIM_STRING/)  = "Win32_SystemServices"
        ...
        GroupComponent (CIM_REFERENCE/ref:Win32_ComputerSystem)*  = <null>
        PartComponent (CIM_REFERENCE/ref:Win32_Service)*  = <null>

        __CLASS (CIM_STRING/)  = "Win32_WMIElementSetting"
        ...
        Element (CIM_REFERENCE/ref:Win32_Service)*  = <null>
        Setting (CIM_REFERENCE/ref:Win32_WMISetting)*  = <null>

        __CLASS (CIM_STRING/)  = "Win32_ServiceSpecificationService"
        ...
        Check (CIM_REFERENCE/ref:Win32_ServiceSpecification)*  = <null>
        Element (CIM_REFERENCE/ref:Win32_Service)*  = <null>
```

With this query, we retrieve the association classes directly referring to the *Win32_Service*, *Win32_SystemServices*, *Win32_WMIElementSetting*, and *Win32_ServiceSpecificationService* classes with their related references, which are *GroupComponent* and *PartComponent*, *Element* and *Setting*, and *Check* and *Element*, respectively.

Keywords of the **Associators Of** and **References Of** statements for schema queries have the same purpose as the keywords of the **Associators Of** and **References Of** statement for data queries. The difference is that they address the classes of the CIM repository instead of the instances created from the classes of the CIM repository. This is the only difference; for the rest, the logic is the same as before. Table 3.4 summarizes the different WQL schema query types with the obtained results.

If you reexamine the data queries and compare some results obtained with the schema queries, the relation between the obtained instances and the obtained classes is not necessarily obvious. You may be confused because associated class definitions retrieved from schema queries are not the same as those retrieved from data queries. The result of performing the data query shown in Sample 3.8 to retrieve the associated instance class definitions of a *Win32_Service* instance lists the following classes:

- *Win32_ComputerSystem*
- *Win32_Service*

The result of performing a schema query of the *Win32_Service* class in Sample 3.26 lists the following classes:

- *Win32_ComputerSystem*
- *Win32_WMISetting*
- *Win32_ServiceSpecification*

The classes listed differ between the data query and the schema query (see Figure 3.5).

Once again, in Chapter 2, when using the **WMI CIM Studio** to understand the dependencies between classes, we used the *Win32_Service* class as an example and saw how the relationships between the different classes were made. We saw that the *Win32_Service* MSExchangeSA instance has no associated instance of the *Win32_WMISetting* and *Win32_ServiceSpecification*. We also saw that the associated instances of some *Win32_Service* instances came from two things: the associations made at the *Win32_BaseService* class (a superclass of the *Win32_Service* class) and the inheritance of the *Win32_DependentService* association class to the subclass. It is for this

Table 3.4 *WQL Schema Queries*

Statement	Keywords						Obtained Result
Select *	From	meta_class	—	—	—	—	All classes.
	From	meta_class	Where	__This	ISA	SourceClass	Any classes made from the *SourceClass*.
Associators Of	—	{*SourceClass*}	—	—	—	—	Nothing is returned
	—	{*SourceClass*}	Where	—	—	SchemaOnly	Any classes where the Association classes associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	AssocClass = *AssocClassName* SchemaOnly	Any classes where the Association classes have a class name matching *AssocClassName* and that are associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	RequiredAssocQualifier = *QualifierName* SchemaOnly	Any classes where the Association classes are using the *QualifierName* and that are associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	RequiredQualifier = *QualifierName* SchemaOnly	Any classes containing the *QualifierName* and that are associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	ResultClass = *ClassName* SchemaOnly	Any classes having a class name matching the *ClassName* and that are associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	ResultRole = *PropertyName* SchemaOnly	Any classes where the role of the classes matches the *PropertyName* and that are associated with the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	Role = *PropertyName* SchemaOnly	Any classes where the role of the *SourceClass* matches the *PropertyName* and that are associated with the *SourceClass*.

Table 3.4 *WQL Schema Queries (continued)*

Statement	Keywords						Obtained Result
References Of	—	{*SourceClass*}	—	—	—	—	Nothing is returned
	—	{*SourceClass*}	Where	—	—	SchemaOnly	Any Association classes been an Association class for the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	RequiredQualifier = *QualifierName* SchemaOnly	Any Association classes containing the *QualifierName* and been an Association class for the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	ResultClass = *ClassName* SchemaOnly	Any Association classes matching the *ClassName* and been an Association class for the *SourceClass*.
	—	{*SourceClass*}	Where	—	—	Role = *PropertyName* SchemaOnly	Any Association classes where the *SourceClass* plays a role matching the *PropertyName*.

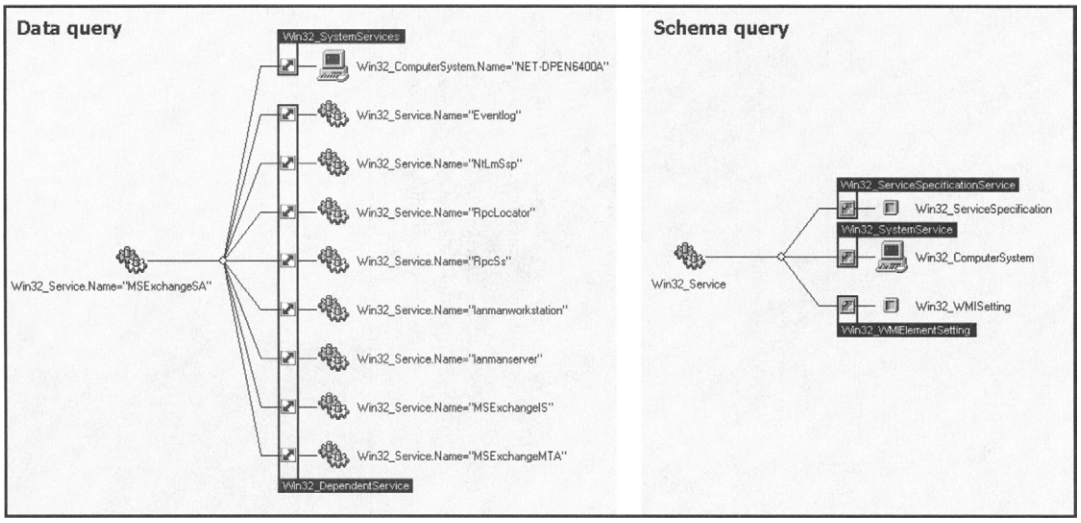

Figure 3.5 *The association differences between a WQL data query and a WQL schema query.*

reason the classes listed differ depending on whether we are examining the schema or the actual instances. It is sometimes quite difficult to see the relationships between classes and instances. Don't hesitate to examine the CIM Schema thoroughly to get a clear understanding of the relationships between classes, between instances, and between classes and instances.

3.2.3 Event queries

Event queries are used to filter events that WMI can return to an application. As we saw in Chapter 2, applications requesting WMI events are called *event consumers*. We briefly described two types of consumers: permanent consumers and temporary consumers. When we practiced the exercise with the permanent *SMTP* event consumer, we used an event query in a MOF file to create the event filter. The *SMTP* event consumer is a permanent consumer because it is registered in the CIM repository as a permanent consumer and is implemented in the form of a .dll. When we script on top of WMI, the scripts that make use of an event query are typically temporary consumers. Whatever the type of consumer we have, the subscription to an event is always performed with a WQL event query. Of course, there are many types of events that an application can receive from WMI. To exploit the capabilities of the WMI events fully, it is necessary to examine these events more closely. Chapter 6 covers WMI event scripting. However, this section provides enough information for you to understand how

WQL event queries work and how we can formulate them for later use in an application or a script.

The best tool to use for event queries without scripting is **WBEMTEST.exe**. As previously mentioned, with an event query an application performs a kind of subscription for a particular event. This means that the application must be active or activated to receive the event notification. **WBEMDump.exe** and **WMI CIM Studio** are not designed for that type of query. This is an important difference between the data queries or the schema queries compared with the event queries. When executing a data query or a schema query, the query produces an immediate result based on the selection criteria. In the context of an event query, the application does not necessarily receive an immediate answer from the submitted query. This is an important difference because to receive a result from an event query, an event matching the submitted query must occur. This forces the application to wait for an undetermined period to get a result. If no event occurs, the application will wait forever if the logic has been written to do so.

To familiarize ourselves with WQL event queries, we will use (for now) only one event type called the *__InstanceModificationEvent*. The *__InstanceModificationEvent* is a system class used to report an instance modification event, which is an event generated when an instance of a class changes in the examined namespace. For example, as we worked with the *Win32_Service* instances, the event reported can be due to the modification of the default startup mode of a Windows service. Changing a Windows service from manual to automatic will be enough to trigger an event if the *Win32_Service* instances of the **Root\CIMv2** namespace match the submitted event query. With the *SMTP* event consumer, the query we used to filter the event was:

```
SELECT * FROM __InstanceModificationEvent WITHIN 10 Where TargetInstance ISA 'Win32_Service'
```

There are many things to say about such a query. First, we recognize the **SELECT** and the **FROM** statements we used in data queries and schema queries. With the **FROM** statement, instead of performing the query on a class (such as a schema query) or on an instance (such as a data query), the query is made on the instance modification event represented by the *__InstanceModificationEvent* system class. Let's examine this query in detail.

3.2.3.1 Polling for events

A new keyword specific to event queries is the **WITHIN** keyword. This keyword, placed before the **WHERE** statement, is not always mandatory in

an event query. Its presence is required if the provider delivering information about the monitored instance is not implemented as an event provider.

Some providers implement a specific interface to inform WMI that an event matching the submitted event query occurred. In such a case, the event provider performs the monitoring and provides an immediate notification in case of modification. When performing event queries on some other instances, there is no event provider developed to monitor the instance modifications. This forces WMI to poll the instances for the provider. It must be clear that the event consumer never performs the poll. Polling is always handled by an event provider or by WMI itself.

If there is no event provider available, at a regular time interval WMI will query the instances (in the CIM repository or via a provider exposing dynamic instances) to determine whether a modification occurred. This means that the notification will be available from WMI at a regular time interval and not when the event occurs. In other words, this interval is the maximum amount of time that can pass before an event notification is delivered. The time specified with the **WITHIN** keyword represents the polling interval in seconds. In the query sample we use, the provider exposing the *Win32_Service* instances has no event provider interface implemented and imposes the use of the **WITHIN** statement. The polling interval can be fractional to deal with values smaller than one second. However, the interval should represent a number of seconds, rather than an extremely small value, because specifying a too-small value can cause WMI to reject the statement as invalid due to the resource-intensive nature of the polling. As a best practice, it is recommended to use an interval greater than five minutes (300 seconds). While doing exercises in this chapter and for academic purposes only, we use a value of two seconds to get faster results.

When using the **WITHIN** statement in such a context, only the instances that have changed during the polling interval will be detected. For example, if the SNMP service is in a running *state* at T (T for time), when we are at $T+1$ (the beginning of a new polling interval), it makes sense that if nothing has changed, there will be no notification. Now, if the SNMP service is stopped and started again during the polling interval (between T and $T+1$), the result will be the same, because the service *state* is not different at $T+1$ compared with its *state* at T. Of course, there was an intermediate change as the service was stopped and restarted, but this intermediate change was made during the polling interval. Because the service was back to its original *state* before $T+1$, no change is detected. Therefore, a notification is created only when the *state* of the instance at T is different at $T+1$. Any intermediate modification (that conducts to the same situation as the

original one) will not be detected. To ensure that intermediate changes are detected, the polling interval must be shorter or an event provider must be available (which does not force you to use **WITHIN** statement and, thereby, triggers an immediate notification).

3.2.3.2 Limiting the scope

The **WHERE** statement has the same purpose in an event query as it has in the context of a data query or a schema query. Although it is used to narrow the scope of the query, its use has some particularities related to the context of an event query. As we saw when working with the *SMTP* event consumer, the *__InstanceModificationEvent* returns two objects embedded in the instance properties—*PreviousInstance* and *TargetInstance*. The *PreviousInstance* property contains a copy of the instance before modification. The *TargetInstance* property contains the new version of the modified instance. The filtering of an event could be executed on the instance contained in the *PreviousInstance* or the *TargetInstance* properties. The usage of the **PreviousInstance** and **TargetInstance** keywords is similar to the usage of the **__THIS** keyword of the schema query. How? In the sense that the **__THIS** keyword identifies the target class for a schema query, and the **PreviousInstance** and **TargetInstance** keywords identify the instance before or after modification for an event query. Both *PreviousInstance* and *TargetInstance* properties are retrieved because the **SELECT** statement uses the "*" filter.

```
SELECT * FROM __InstanceModificationEvent WITHIN 2 Where TargetInstance ISA 'Win32_Service'
```

It is possible to formulate the query to retrieve only the *TargetInstance* of the *__InstanceModificationEvent* instance:

```
SELECT TargetInstance FROM __InstanceModificationEvent WITHIN 2
                                                    Where TargetInstance ISA 'Win32_Service'
```

The **ISA** operator is a WQL-specific operator that can also be used in event queries. When **ISA** is included in the **WHERE** statement of an event query, it requests notification of events for all classes within a class hierarchy rather than a specific event class. Therefore, its use in an event query is very similar to its use in a schema query. Let's try this WQL event query by using the **WBEMTEST.exe** tool "Notification Query" window. Make sure that you use asynchronous notification to get the best performance. Again, later we see the different notification types available.

Once the query has been coded in **WBEMTEST.exe** GUI and applied as shown in Figure 3.6, any modification of a Windows service instance will generate an event. For example, start the SNMP service to see what happens.

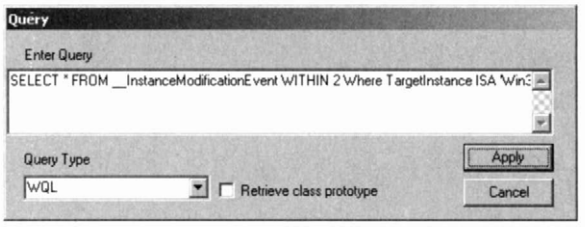

Figure 3.6 *Entering the event query in WBEMTEST.exe.*

Once the SNMP service starts, the query result window, which appears as soon as you submit the query, contains an instance of the *__InstanceModificationEvent* (see Figure 3.7). By clicking twice on this event, it is possible to examine the *__InstanceModificationEvent* instance (see Figure 3.8). The **WBEMTEST.exe** tool displays two embedded objects in the *PreviousInstance* and *TargetInstance* properties.

By double-clicking each of these objects, you can see the properties of the instances contained in the *PreviousInstance* (Figure 3.9) and the *TargetInstance* properties (Figure 3.10) when starting the SNMP service.

3.2.3.3 Limiting the scope with data selection

Just as with data queries, it is possible to combine some logical operators to add some selection criteria. The event query we used reacts to any *Win32_Service* instance modification. If we want to receive event notification only when the "SNMP" service is stopped, the query is as follows:

```
SELECT * FROM __InstanceModificationEvent WITHIN 2 Where
                                          TargetInstance ISA 'Win32_Service' And
                                          TargetInstance.Name='SNMP' And
                                          TargetInstance.State='Stopped'
```

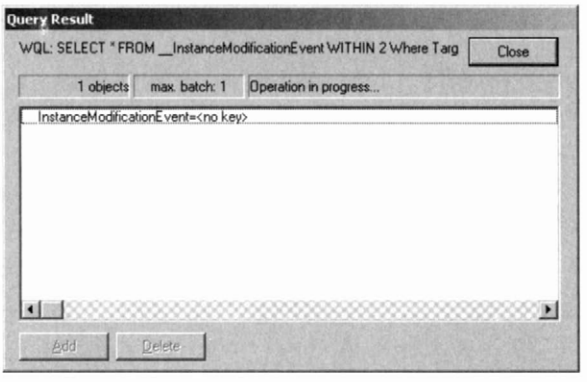

Figure 3.7 *The WBEMTEST.exe query result window for event queries.*

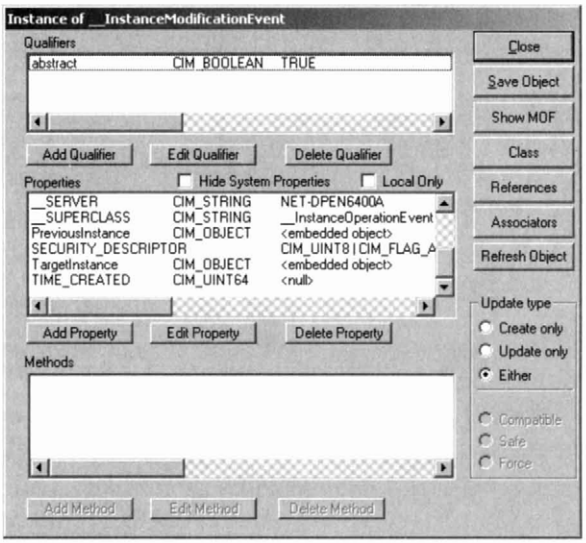

Figure 3.8 *Examining the event query result with WBEMTEST.exe.*

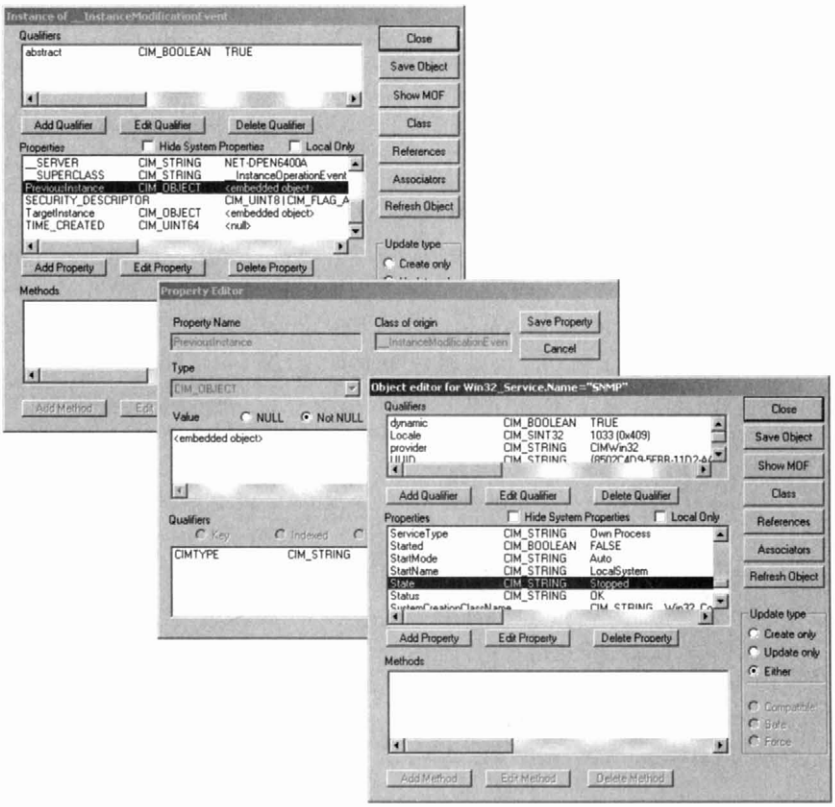

Figure 3.9 *Viewing the PreviousInstance result with WBEMTEST.exe.*

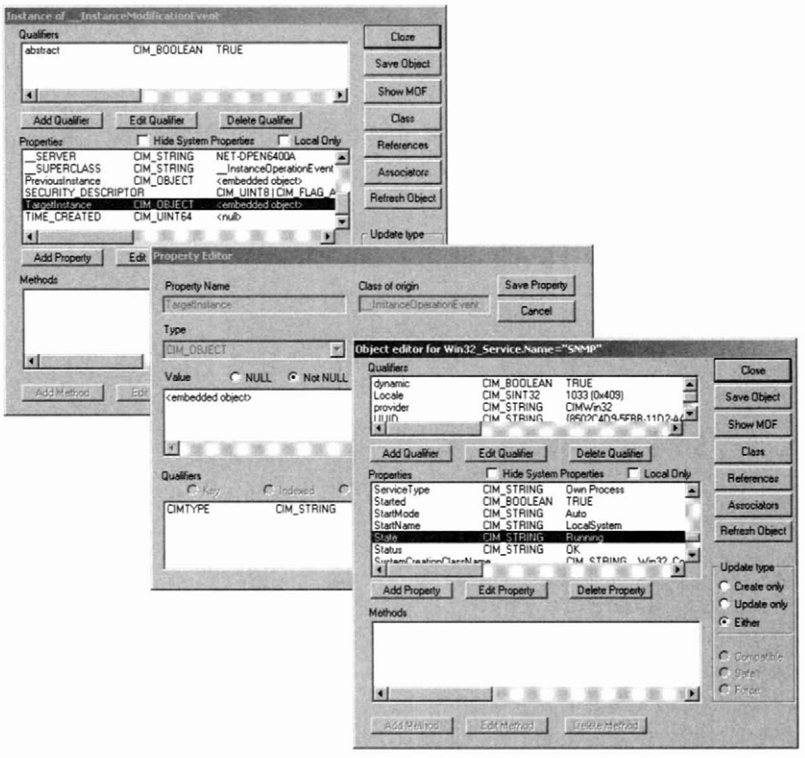

Figure 3.10 *Viewing the TargetInstance result with WBEMTEST.exe.*

3.2.3.4 *Limiting the scope by grouping events*

The last query narrows the scope based on a matching data set related to the object instance. Using the data instance is not the only method for narrowing an event query scope. With event queries, it is also possible to narrow the scope in time. For this the **WITHIN** and the **HAVING** statements can be used. In this context, the **WITHIN** statement is used differently (see the **WITHIN** statement for the polling interval in Section 3.2.3.1) and in combination with the **GROUP** statement. These statements are used to protect event consumers from being flooded with notifications that occur too frequently. Consumers that do not require notification each time an event occurs can use these statements.

The **GROUP** statement causes WMI to generate a single notification to represent a group of events. Previously, the obtained result was an instance of the *__InstanceModificationEvent* for each event. With the **GROUP** statement, the obtained result is an instance of the *__AggregateEvent* for a group of events. We will not delve into the details of such a system class instance

in this chapter, because we will focus on the events in the WMI events scripting in Chapter 6. However, we can say that the *__AggregateEvent* instance contains an object that is representative of the events that occurred in the group. This statement concentrates on a group of events, and since event notifications occur over an undefined period of time, grouping events implies the definition of an interval. This period of time is known as the *grouping interval* and is defined in seconds with the **WITHIN** statement. This is why the **WITHIN** statement is always used in combination with the **GROUP** statement. Let's take a sample:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                         TargetInstance.State = 'Stopped'
                                                  GROUP WITHIN 60
```

It is important to note that the **WITHIN** statement is referenced twice in this WQL query. The first **WITHIN** statement sets the polling frequency to two seconds, because there is no event provider for the *Win32_Service* instances. You will notice that we use a shorter interval period to make sure that we detect any service change to the "stopped" *state* fast enough. Now, let's talk about the second **WITHIN** statement by examining the following query first:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                         TargetInstance.State = 'Stopped'
```

With this query, if we start and stop the SNMP service three times, we will generate several events. If we stop the service to get a "stopped" *state* when the polling interval terminates and on the condition that the service *state* was "started" at the beginning of the same polling interval, we generate the same event three times. By adding the **GROUP WITHIN 60** statement, we will receive only one event grouping all the changes of the same nature performed during a period of 60 seconds. Therefore, even if the detection of the "stopped" *state* of the service was made three times, the grouping interval of 60 seconds will create one notification of the *__AggregateEvent* instance instead of three notifications of the *__InstanceModificationEvent* instance. Figure 3.11 shows the *__AggregateEvent* instance received, and Figure 3.12 shows the content of the *__AggregateEvent* instance.

The last window in Figure 3.12 shows that we obtain an *__InstanceModificationEvent* instance embedded in the *__AggregateEvent* instance. From the *__InstanceModificationEvent* instance, we see the *PreviousInstance* and *TargetInstance* properties. From this point, we are examining an object similar to the one obtained in Figure 3.10. Like the *__InstanceModificationEvent* instance, the *__AggregateEvent* exposes two

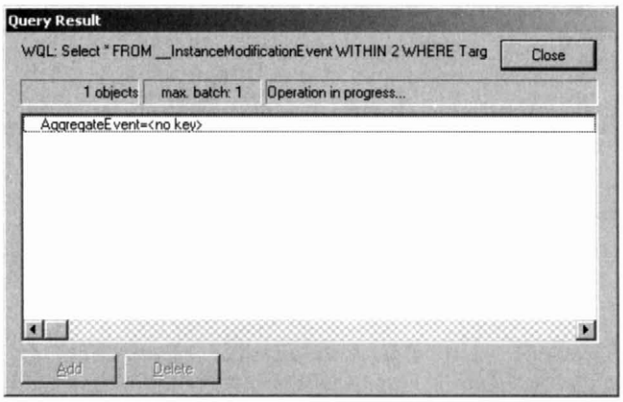

Figure 3.11 *The WBEMTEST.exe query result window for aggregated event queries.*

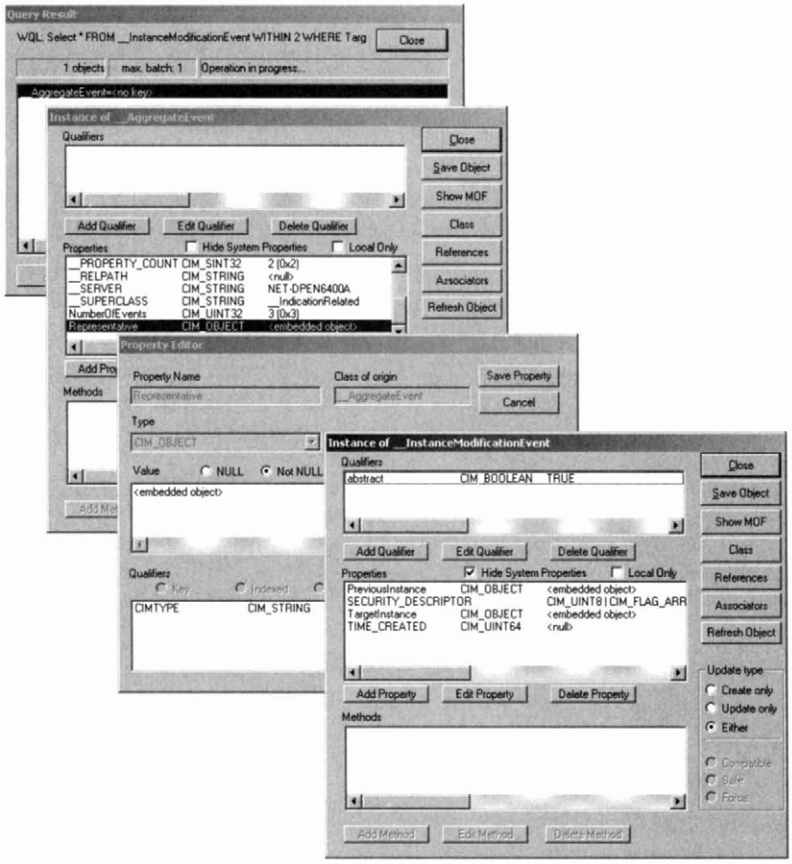

Figure 3.12 *Examining the aggregated event query result with WBEMTEST.exe.*

specific properties called *NumberOfEvents* and *Representative*. Like the *PreviousInstance* and the *TargetInstance* properties, the *Representative* property contains an embedded object. The *NumberOfEvents* property contains the number of times the event occurred for the *__AggregateEvent* instance.

3.2.3.5 Limiting the scope with a minimum of occurrences in a group

This is where the **HAVING** statement is used. The previous event query triggers one notification if any *Win32_Service* instance is stopped one or more times during a period of 60 seconds. For example, it would be interesting to get an event notification only if five occurrences of the same event occur during the same period of time. Any other similar event generated less than five times in the same grouping interval will not trigger a notification. In this case, the WQL event query is as follows:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                         TargetInstance.State = 'Stopped'
                                                  GROUP WITHIN 60 HAVING NumberOfEvents>=5
```

This means that if a *Win32_Service* instance is detected as "stopped," with a polling interval of two seconds at least five times during a 60-second period, a notification event will be sent in the form of an *__AggregateEvent* instance containing a representative instance of the modified instances. This form of query particularly suits the event log monitoring to track specific message creations. Moreover, the WMI provider that gives access to the event log is also implemented as an event provider, which eases the notification of new event log message creations, given that the **WITHIN** statement for the polling technique is no longer necessary.

3.2.3.6 Criteria to group events

In the previous queries, because we changed only the SNMP service state, it was easy to get in the *__AggregateEvent* instance a representative *__InstanceModificationEvent* instance of the SNMP service, as it was always the same service. But what happens if we stop the SNMPTRAP service during the same period? Let's try this with the following query:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                         TargetInstance.State = 'Stopped'
                                                         GROUP WITHIN 60
```

Notice that we removed the **HAVING** statement to eliminate the required number of changes for the "stopped" *state* during the grouping interval. Next, as we have two Windows services to stop (SNMP and SNMPTRAP), we will use a very small command file to ease our work:

```
@ECHO OFF
IF .%1.==.. GOTO END
IF .%1.==.0. GOTO END

SET /A I=%1
SET /A C=0

:LOOP
SET /A C=%C%+1
ECHO -------------- Loop number %C% --------------

CALL :STOPSVC SNMP
CALL :STOPSVC SNMPTRAP

IF NOT .%C%.==.%I%. GOTO LOOP
GOTO END

:STOPSVC
NET START %1
NET STOP %1
GOTO :EOF


GOTO :EOF
:END
```

This command file takes a number as a parameter. This number represents the number of times that the SNMP and SNMPTRAP services must be stopped. For this, the command file executes a loop until the given number is reached. This command file is a pretty easy one and will be a great help for simulating the expected behavior of the WQL event query. So, if we execute this command file with the parameter "2," we will stop the SNMP and SNMPTRAP services twice. The command file output will be

```
C:\>Stop 2
-------------- Loop number 1 --------------
The SNMP Service service is starting.
The SNMP Service service was started successfully.

The SNMP Service service is stopping...
The SNMP Service service was stopped successfully.

The SNMP Trap Service service is starting.
The SNMP Trap Service service was started successfully.

The SNMP Trap Service service is stopping.
The SNMP Trap Service service was stopped successfully.

-------------- Loop number 2 --------------
The SNMP Service service is starting.
The SNMP Service service was started successfully.
```

```
The SNMP Service service is stopping..
The SNMP Service service was stopped successfully.

The SNMP Trap Service service is starting.
The SNMP Trap Service service was started successfully.

The SNMP Trap Service service is stopping.
The SNMP Trap Service service was stopped successfully.
```

Now, if we look in the **WBEMTEST.exe** result query window (see Figure 3.13), we see that we get one *__AggregateEvent* instance.

As we stopped two different services, what does the *__AggregateEvent* instance contain? First, the *NumberOfEvents* property contains the value

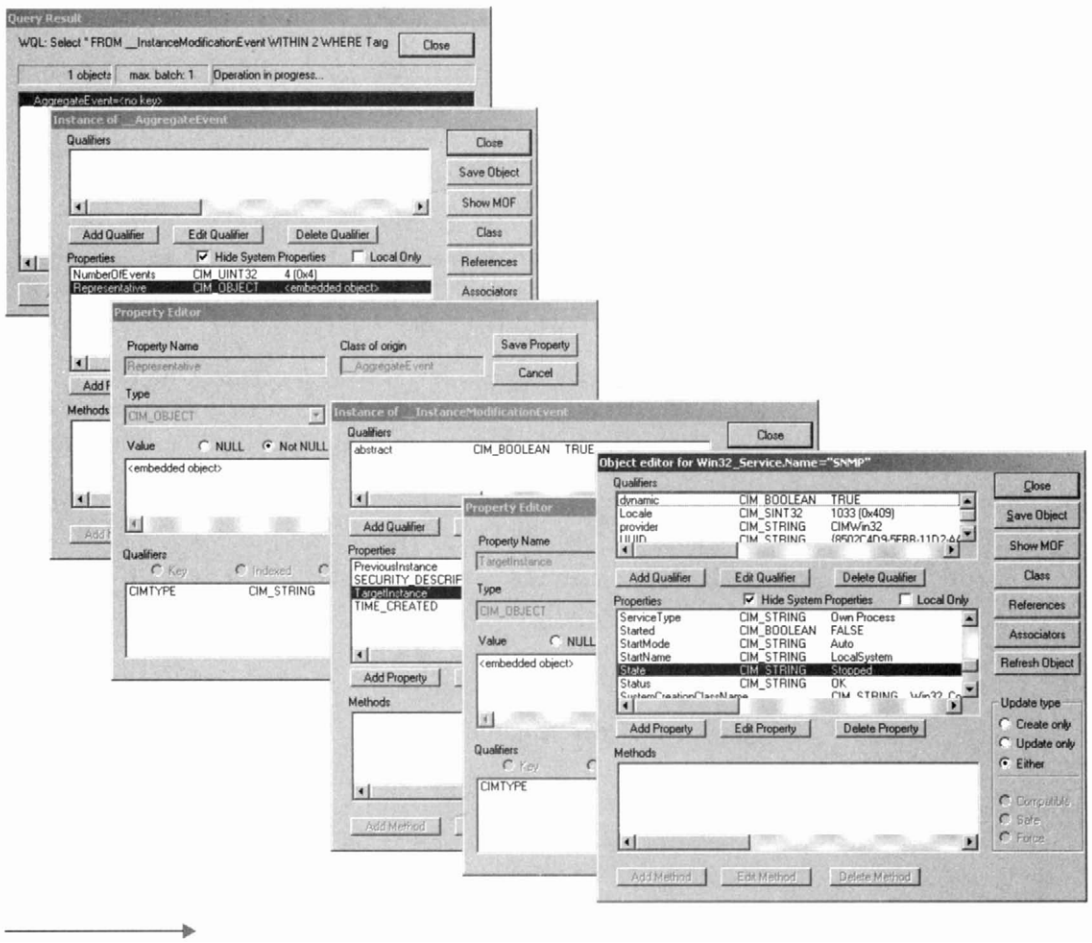

Figure 3.13 *Examining the aggregated event query result with WBEMTEST.exe when two different services are stopped.*

4—exactly the number of Windows services stopped (twice for the two services). Now, if we look at the *Representative* property, we find an instance representing the SNMP service. What about the SNMPTRAP service? Well, this modified instance is simply not represented. Now, with the command file, we stop the SNMP service first, and the SNMPTRAP service next. If we invert the order, the *Representative* property will contain the SNMPTRAP service and not the SNMP service. In any case, we always lose one modified instance. This is where the **BY** statement is useful. The BY statement allows the grouping of events by criteria. Let's modify the previous query as follows:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                        TargetInstance.State = 'Stopped'
                                                  GROUP WITHIN 60
                                                  BY TargetInstance.Name
```

If we submit this query and restart the command file with the same value, we execute the same modification on the two Windows service instances, but the obtained result is different (see Figure 3.14).

Here we are! Now, we get two *__AggregateEvent* instances. Because the **BY** statement groups each event on the *Name* property of the object embedded in the *__InstanceModificationEvent* instance, we get two different event instances for each stopped service. If we stop a third service during the grouping interval, we will get three *__AggregateEvent* instances.

We can add the *NumberOfEvents* filter used before and, with the help of the command file, we can try the following query:

```
Select * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_Service' And
                                                        TargetInstance.State = 'Stopped'
                                                  GROUP WITHIN 120
                                                  BY TargetInstance.Name HAVING NumberOfEvents>=3
```

Notice that we extended the grouping interval to make sure that we can stop the Windows services at least three times. This query will produce a number of notification events equal to the number of Windows services stopped at least three times in a period of 120 seconds.

The formal syntax representation of an event query is as follows:

```
SELECT * FROM EventClass [WITHIN polling_interval] WHERE property operator value
         [GROUP WITHIN grouping_interval] [BY property_list]
         [HAVING NumberOfEvents operator integer]
```

When using the **GROUP** statement, the **WHERE**, **BY**, and **HAVING** statements are optional. From a semantic point of view, the **WHERE** statement is also optional, but must be specified from an operational point of view to get a match on an event class representing the modified instance.

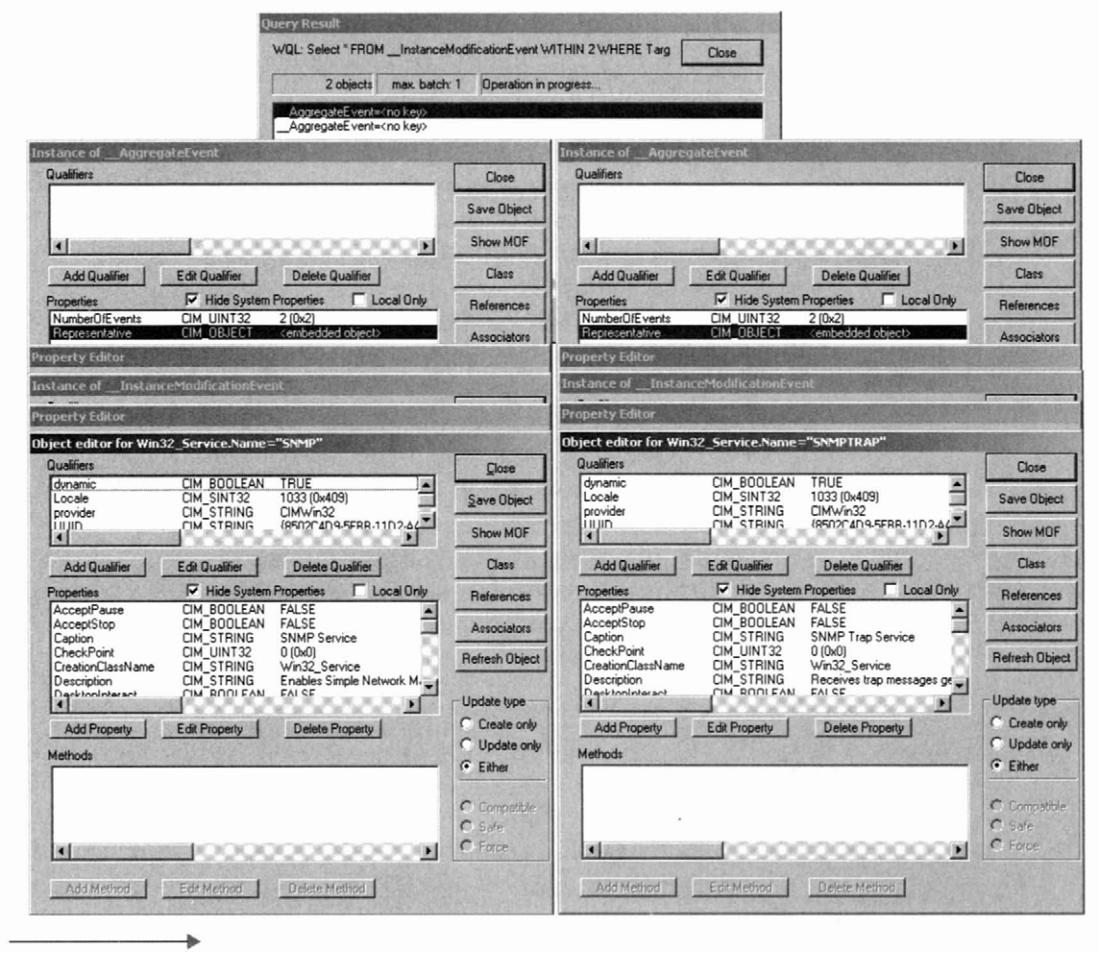

Figure 3.14 *The effect of the BY statement on the aggregated event query results when two different services are stopped.*

3.3 Extended WQL

The Extended WMI Query Language is provided with the installation of Microsoft Systems Management Server (SMS). Since we do not talk about SMS in this book, we just mention the existence of Extended WQL to clarify its positioning with regard to WQL. Extended WQL supports a superset of WQL. Extended WQL is like WQL in that it is a retrieval-only language used to create queries. It cannot be used to create, modify, or delete classes or instances. Extended WQL, as with WQL, is based on the ANSI SQL standard, but includes a much broader range of operations than WQL,

especially oriented for the use of SMS. Extended WQL comes with SMS Service Pack 2, is fully case insensitive, and supports the standard WQL comparison operators, plus the **LIKE** and **IN** operators. For those of you working with SMS, please refer to the SMS documentation for more information about Extended WQL.

3.4 Summary

Throughout this chapter, we reviewed the possibilities of WQL. As WQL is one of the key elements required to work with WMI, it is a valuable study for the discoveries we make in the subsequent chapters. In most cases, WQL is a primary input to execute WMI tasks. The difficulty in learning WQL is that we have to work with some WMI elements that are still unknown at this stage, but are examined later in the book. However, because WMI is a mix of various features interacting with each other, it is necessary to start from somewhere. When we dive deeper into the elements constituting WMI, the interactions between various elements will become clearer.

Usually, when a script writer starts to develop on top of a COM abstraction layer, the key element is to master the object model used, such as Active Directory Service Interfaces (ADSI), Collaborative Data Objects (CDO), or CDO for Exchange Management (CDOEXM). With WMI, the learning phase starts at the level of the CIM repository with WQL. Learning the CIM Object Model is necessary in order to understand the kind of information available from a WMI system. Learning WQL is essential to be able to formulate inputs to gather data from the CIM repository. When working with WMI event scripting, WQL is the primary input to formulate event queries. Of course, being able to script with the WMI events and learning the scripting interfaces exposed by WMI is another learning process, which is independent of the CIM repository and the knowledge of WQL. At this stage, we are ready to start the discussion of the WMI scripting interfaces, which leads us to the next chapter.