Chapter 2

WMI: Introduction

Many of you will probably not have heard of WMI (Windows Management Instrumentation) before or if you have there's a good chance you know very little about it. WMI is Microsoft's primary management enabling technology for Windows so you would think more people would have heard about it. Some say that WMI is Microsoft's best kept secret, so you're not alone when it comes to not knowing much about it. By the end of this and the next Chapter, you'll have gained enough information to know why knowing this secret is important. If you think that this topic doesn't really affect you, then that's only because you haven't read the rest of the Chapter. Although WMI is commonly used by Administrators to monitor both local and remote machines using scripts, developers also need to know about WMI as well. Sometimes it can be the only place where you can find a piece of information, or it could significantly reduce the amount of code you need to perform a given task. You also need to know about WMI if you want your applications to be monitored in production by third party applications or those Administrators and their scripts. If you have used WMI before you will have probably used the COM based API's; the .NET equivalent should prove to be easier to use.

WMI 101

WMI has been around since 1998 (appears Microsoft can be good at keeping secrets sometimes) when it was released as part of the Windows NT 4.0 Service Pack 4. Since then it has been incorporated into Windows 2000, Windows XP and Windows .NET Server and forms the core management technology in these systems. It is also available for download at the Microsoft Web site for Windows 9x operating systems. WMI is Microsoft's implementation of the Web-Based Enterprise Management (WBEM) specification, which is an industry initiative overseen by the Distributed Management Task Force (DMTF) to develop a standard technology for accessing management information in an enterprise environment. The standard was used to address the ability to access, configure, manage and monitor system resources. The result is that using the Microsoft implementation of WBEM (WMI) you can perform these tasks with just about every Windows resource. You'll get a better understanding of what this really means as you work your way through the Chapter. Also, you may have looked ahead and noticed that WMI is mentioned in many of the Chapters that follow. This is because you can use WMI or the managed classes to do many of the same things. There are advantages and disadvantages to each, but I will explore both so you can make the correct choice when the time comes. However, that is later; first you need to understand what WMI is and how it works so you can have a better understanding of the next few Chapters.

Tip: WMI is based on the WBEM specification; this means WMI is not a Microsoft specific technology. Searching the internet for WBEM will discover other implementations for other Operating Systems such as Linux.

Terminology

Having just covered a bit of the history; let's now look at the more technical parts of WMI. Given that WMI is fairly new to a lot of you, I'll start by defining a number of terms that you'll need to understand to make proper use of this technology.

* **Provider:** Acts as an intermediary between WMI and a managed resource, such as an Event Log. Providers request information from, and send instructions to the WMI managed resource on behalf of consumer applications and scripts.
* **Consumer:** Is a script, enterprise management application or any other application that makes use of WMI resources.
* **CIM:** The Common Information Model is a schema, also known as the object repository or class store that models the managed environment and defines every piece of data exposed by WMI.
* **CIMOM:** CIM Object Manager (pronounced see-mom) handles the interaction between consumers and providers. The term originates from the WBEM specification maintained by the DMTF. You can think of CIMOM as the WMI information broker in that all WMI requests and data go through the CIMOM.
* **Class:** A class is a template for a discrete WMI managed resource, and all instances of the resource use this template. Classes represent the things computers have such as disks, event logs, files, folders, memory, printers, processes, processors and so on. Classes can be thought of in the same way as an abstract class in .NET in that it provides the interface or schema for a class instance.
* **Namespaces:** CIM classes are organized into namespaces. Much like you partition your classes into namespaces in .NET, the CIM also uses this type of mechanism to control the scope and visibility of managed resource class definitions. Note that although the concept is the same as .NET class namespaces their implementations are not related.
* **Events:** WMI allows you to register for events which will notify you when any WMI object is created, deleted or modified.
* **Remotable:** Allows access to objects which are stored on other machines. Microsoft chose DCOM for this purpose, although the performance of standard DCOM produced a lot of excess overhead. To improve performance, Microsoft developed sophisticated custom proxies and stubs to compress information and minimize this overhead. These proxies and stubs are transparent to the developer, but they are responsible for the awesome performance of remote connections.

WMI can be used to build tools that organize and manage system information so that administrators or system managers can monitor system activities more closely. As an example, you can develop an application using WMI that pages an administrator when a

Web server crashes, or have a badly behaving process terminated if it exceeds more than 90% of CPU usage. Microsoft describes WMI in their documentation in the following way:

> ***Windows Management Instrumentation (WMI) is a scalable system management infrastructure that uses a single consistent, standards-based, extensible, object-oriented interface. WMI provides you with a standard way to interact with system management information and the underlying WMI APIs. WMI is used primarily by system management application developers and administrators to access and manipulate system management information.***

Why use WMI?

Given that you're knowledge about WMI may be limited a good question to start with would be why use it? WMI essentially wraps a lot of what already exists, so why would you want to have yet another layer on top of the existing APIs. There are a number of good reasons to make use of the power that WMI offers whether you're an administrator or developer. WMI allows access to an enormous amount of information about both local and remote machines. If you need to know something about the system and you can't find a managed class that provides the information, then there's a high probability that WMI will come to your rescue. To give you some idea, if you are running Windows 2000 you would have approximately 600 classes and with Windows XP approximately 900. The number of classes will depend on the version of Windows you're running and the software you have installed. As you'll learn later, WMI provides all this information in a standard way. How many times have you either had trouble finding a particular API to provide the information you wanted or couldn't quite work out how to use it when you did. APIs in general have been poorly documented; WMI has much better documentation and is in general much easier to understand. The most important aspect however is that there is an enormous amount of information ready to be used. Some of the key reasons why you would use WMI are:

* **Industry Standard:** Uses an industry standard class-based schema. The benefit here is that the classes can be very precise in their definition and this allows others to easily understand and use your classes. This also aids in the Administration side as the schema is also discoverable.
* **Simple Object Model:** Easy access to management objects through either a scripting interface or through a programming language such as VB.NET, Visual Basic 6, Windows Scripting, C# or C++.
* **Query Language:** WMI provides an infrastructure for executing WQL (WMI Query Language) queries. If a provider doesn't support query optimization, WMI will provide it for free. The query language is based on SQL and so is also easy to learn.
* **Easier deployment:** When developing DCOM solutions you need a proxy-stub DLL to be installed on each of the client machines. By contrast accessing WMI management objects requires no additional installation on client machines, assuming WMI is available.

* **Extensibility:** The model is designed to allow new provider modules which supply data and events to be plugged in. Some of the existing provider modules that are currently available include Microsoft Internet Information Server (IIS), SQL Server, Active Directory, Event Logging and Performance Counters. This is by no means a definitive list as just about every system related resource has a provider in Windows. Also, it doesn't end with Microsoft provided providers. Many ISVs are also providing WMI enabled applications which can be managed by any WMI aware application.
* **Remotable:** An extremely powerful feature of WMI is that it is remotable using DCOM and custom marshaling for improved performance. This is a particularly good feature as many otherwise equivalent framework classes that provide the same data don't provide this ability.
* **Events:** WMI provides a high performance and extensible event mechanism which a consumer of the event can use to be notified when any WMI object is created, deleted or modified. For example if you want to know when the CPU usage becomes greater than 85%, you could get WMI to notify you via an event. You can also define permanent consumers which are components that are invoked whenever a specified event occurs. WMI also comes with a set of standard consumers that perform tasks such as running scripts, creating events in the event log or sending an email upon receiving a WMI event.

What can I do with WMI?

Using WMI, an application can detect and discover management information such as what operating system (including the service pack number) is installed on a server. This can be either the local machine or one locked away in a remote location. An application can also determine how many processors the computer system has, which services are running in a process, which network connections are currently open, and which domains have a trust relationship with the local domain of the host computer.

With the power that WMI provides you can perform actions against management objects such as stopping or starting a process or a service, deleting all disk drive shares, recursively unloading all running Web applications from a specific IIS virtual directory, and other dangerous things. The good news is that WMI performs these actions on behalf of the client, so you will be able to do only as much the client context permissions allow.

Using WMI, an application can also receive management event notifications for such things as when a specified service stops, when the amount of free space on a system disk drops below a certain limit (I wish more IT departments used this one!), when a process uses more than a specified amount of virtual memory, when an application has encountered a significant error condition and raised a WMI event, and so on. As mentioned earlier WMI is not restricted to just system resources. Applications can also provide WMI services; SQL Server is one such application, which allows you to be notified if a table changes or has data added, deleted or modified. To list all the things you can possibly do with WMI would take a book in itself; however, I would start checking

the applications you interface with and discover what their WMI capabilities are, because you may be surprised what they will let you do.

What are my Options?

When it comes to your options or what you can do with WMI, it would almost be easier to try and describe what you can't do as opposed to what you can do. There are a few different options however for actually programming WMI itself which I'll detail below. I'll also cover the more common providers available for WMI, but again I can't stress enough that this is only scratching the surface of what information lies within the CIM.

* **Managed Classes:** Using the classes found in the System.Management namespace you can access any WMI class. There are classes for retrieving class data and also for hooking into WMI events. The model is also said to be easier than its COM counterpart, as it has many specific class types.
* **Server Explorer:** Microsoft introduced a Beta version of their WMI snap-in for VS.NET 2002 and then released a version for VS.NET 2003 (a.k.a. Everett) shortly after its release. It is highly recommended that you download this plug-in for your version of VS.NET. See the resources section for the URL details.

Table 2-1 – Sample of WMI Providers

Provider	Namespace	Description
Active Directory	root\directory\ldap	Active Directory is the Windows directory service that provides a unified view of complex networks.
Event Log	root\cimv2	Allows access to Windows Event logs. Supported features include: read, backup, clear, copy, delete, monitor, rename compress, uncompress and event log settings.
Performance Counters	root\cimv2	Provides access to performance data. These can be both system supplied or your own custom counters.
Registry	root\default	Able to read, write enumerate, monitor, create and delete registry keys and values.
SNMP	root\snmp	Provides access to SNMP MIB Data and traps from SNMP managed devices.
Win32	root\cimv2	This is the core namespace and provides access to computers, disks, peripheral devices, files, folders, networking, operating system, printers, processes, security and so much more!
Windows Installer	root\cimv2	Provides access to information about what has been installed on a system.

From Table 2-1 you can see there is a large range of information that can be accessed using WMI, which prior to WMI (remembering WMI is a pre .NET technology) would have required extensive knowledge of the Win32 API. Now it is fairly easy to access this

information from both .NET and from the older COM based API. For more information about the COM based version of the API see the resources section at the end of the Chapter.

Choosing the correct option

I have detailed the different options available, however how do you know which options to use in any given situation? This section will discuss each of the available options, and when you might like to make use of them. They are of course only suggestions, and you will ultimately need to consider your applications overall architecture when making a decision.

Managed Classes

If you are going to be accessing WMI from within .NET you will ultimately be using the managed classes provided in the System.Management namespace. The main choice you have hear is; do I make use of the Server Explorer (see below) or code directly against the frameworks classes. The 'problem' with the managed classes is that they are late bound in that they use the same object for any class type. This means you can introduce errors into your code if you are not careful. It also means IntelliSense doesn't get a look in either; you'll need to know the name of the properties you want to access on a given CIM class.

Like anything that is late bound you can create your own wrappers for any given class that you use regularly which removes the disadvantages just mentioned. However, it also means you have a bit of code to write and you'll loose the generic nature that WMI provides.

My personal preference is to go with wrappers where it suits, but given the power that WMI provides you'll probably find it difficult to create a wrapper that is intuitive yet still retains all the power of WMI. There will therefore have to be a compromise in this respect. Later in the book I show you how to create a fairly complex wrapper for the Event Log class, which will demonstrate many of the advantages that WMI provides.

Server Explorer

As mentioned previously Microsoft developed WMI Extensions for Visual Studio.NET 2002 and 2003. These extensions which are covered in detail later allow you to easily attach to WMI classes without knowing very much about WMI. Obviously, the more you know about WMI the better your use of the extensions will be, however for many simpler tasks, very little WMI knowledge is required. The extensions also allow you to drag and drop resources onto a Form in a similar manner to any other resource managed by Server Explorer (see Chapter 1). As will be covered later this produces a new class in your project which acts as a wrapper for the resource you've selected. Although the amount of source code produced can be quite large, it does simplify the development of WMI applications. Therefore as a recommendation I would make is to make use of the extensions to the extent that you feel they are required. You may decide to create your

own wrapper classes rather than rely on the ones produced by Microsoft; however their classes appear to be quite comprehensive for a given resource.

Installing WMI for VS.NET

Unfortunately Microsoft didn't make the extensions part of VS.NET 2003, so you'll still need to download and install it; this applies to VS.NET 2002 also. You can download the beta or RTM version from the URL in the resources section at the end of the Chapter. Once you've downloaded the setup file, ensure your VS.NET is closed then you only need to run the msi file and your done!

How to make use of these extensions will be covered in detail in the WMI Server Explorer Extension section in the next Chapter.

WMI Architecture

The WMI architecture consists of three core layers with a few sub layers as shown in Figure 2-1. I will discuss each of these layers starting at the bottom and working our way up. The top layer being the application you will be coding.

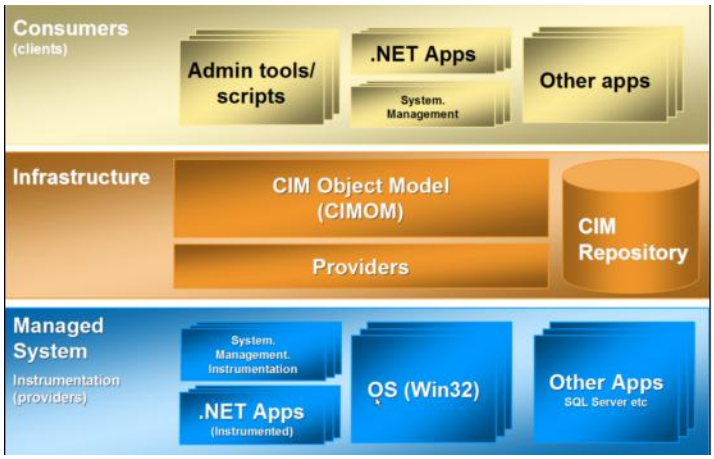

Figure 2-1. WMI Architecture

Managed System

The managed system consists of any logical or physical component, that is exposed and manageable using WMI. Examples of such resources would be the computer system, event logs, files, disks, peripherals, printers, security settings, Active Directory, Windows Installer and Windows Driver Model (WDM) device drivers to name only a few. These resources communicate with WMI via a provider which is covered next.

Provider

As mentioned previously a provider acts as the intermediary between WMI and the managed system. Providers request and send information to WMI managed resources on behalf of a consumer request. The providers do a good job of hiding the implementation details from the consumer. This is achieved by exposing the resource to WMI based on the standards based access model. The providers communicate with the managed resource via the resources native Win32 APIs, and communicate with CIMON using WMI programming interfaces.

Although there are many built in providers which are part of the later versions of windows (see Table 2-1), developers can also build their own custom providers. I will cover how to do this in the next Chapter.

CIM Object Manager

As previously mentioned CIMON is the information broker which all requests for data flow though. Apart from providing the common interface through which consumers access WMI, the CIMON provides a set of core services.

* **Event processing:** Allows a consumer to subscribe to events that represent a change to a WMI-managed resource. For example, you can subscribe to an event indicating when the amount of space on a logical disk drive drops below an acceptable threshold. The CIMOM polls the managed resource at an interval you specify, and generates an event notification when the subscription is satisfied.
* **Provider registration:** WMI providers register location and capability information with the CIMOM. This information is then stored in the CIM repository.
* **Query processing:** Allows a consumer to issue queries against any WMI-managed resource using the WMI Query Language (WQL). For example, you can query the event logs for all events matching a specific Event ID, which occurred during the past 24 hours. The CIMOM performs the evaluation of the query in cases where providers don't natively support query operations. In these cases the performance may be slow if the provider contains a lot of information.
* **Remote access:** Consumers access remote WMI-enabled systems by connecting to the CIMOM on the remote system. Once a connection is established, consumers can perform the same operations that can be performed locally.
* **Request routing:** The CIMOM uses the provider registration information to route a consumer.
* **Security:** The CIMOM controls access to WMI-managed resources by validating each user's access token before the user is permitted to connect to WMI, on either the local computer or a remote computer. WMI does not override or circumvent security provided by the operating system.

Consumers of WMI make use of the services provided by the CIMON to mine data, subscribe to events, or to perform some other management related task. The CIMON also acts as the wrapper around the CIM Repository which is covered next.

CIM Repository

The CIM is based on the DMTF Common Information Model Standard schema. This schema allows a uniform representation of management information from various sources. The CIM is also built upon the notion of classes that represent objects created and stored in the repository. Most of the classes that are stored in the CIM act only as templates for the providers. This allows you to access the class definitions, but no actual instance data is stored within the CIM. There are a few classes which do however store some data within the CIM, but these are the exception rather than the rule. This architecture allows the most accurate information to be provided to the consumer of the resource as it is supplied by the underlying provider at the time it is requested.

Namespaces

All CIM classes are organized into namespaces. In the same way you can use namespaces in your .NET applications to organize and control the scope and visibility of your assembly; so does the CIM. Although the concept is the same, its implementation has no connection to that used in .NET. The namespaces are grouped by related classes representing a specific technology or area of management. As with any namespace implementation, each class within a namespace must be unique. However, unlike some implementations, classes from one namespace cannot be derived from a class in another namespace.

The most common namespace you will deal with is root\cimv2 (the default) although it will not be the only one you will need to deal with. For example if you wanted to access the registry you would need to use root\default; know one seems to know why it's in this namespace, given all the other system related resources are in root\cimv2.

It's important to understand namespaces because when you connect to WMI, you also connect to a default namespace if you don't supply one. This is important because if you try to access a class from a different namespace you will get an error. To ensure that you are always working with the correct namespace you should specifically define the namespace you are interested in.

> **Note:** I will show you how to build a query tool in the next section, which will be used to test your queries. This tool will allow you to run your queries easily and enable you to test any data query. A reference to the tool within any code snippets informs you of the item in the dropdown list which relates to the query. You'll also see a reference to Result which can be either Summary or Detail; this will become clearer when we build the tool.

The following code which was taken from the QueryTool project you will build later demonstrates how you would connect to a specific computer and namespace; the details of which will be covered a little later:

```
Dim objScope As ManagementScope
objScope = New ("\\" & Me.txtMachine.Text & "\" & Me.txtDefaultNamespace.Text)
```

You then supply this object when you execute your queries:

```
Dim objWMISearcher As ManagementObjectSearcher
objWMISearcher = New (objScope, New ObjectQuery(Me.txtQuery.Text))
```

> Note: The standard default namespace is a registry setting and can therefore be changed. This may result in your code no longer working as expected. It is therefore recommended that you always explicitly define the default namespace you are working with. The registry key is:
>
> *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\Scripting\Default Namespace*.

If you wanted to know what the current default namespace for your WMI service is, you can write a simple WQL query to find out:

```
' QueryTool: Namespaces: Default
SELECT ASPScriptDefaultNamespace FROM Win32_WMISetting

' Result (Detail):
ASPScriptDefaultNamespace: root\cimv2
```

You can also find out what namespaces are available, this can be useful if you want to know if a machine has a particular namespace installed. For example if you were running Windows .NET Server you would see the namespace root\MicrosoftDNS, however if you were running Windows XP as I am, you won't. You can try this yourself using the following query:

```
' QueryTool: Namespaces
SELECT * FROM __NAMESPACE

' Result (Summary):
\\INSPIRON8200\ROOT:__NAMESPACE.Name="SECURITY"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="RSOP"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="Cli"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="WMI"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="CIMV2"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="MSAPPS10"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="Policy"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="Microsoft"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="DEFAULT"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="directory"
\\INSPIRON8200\ROOT:__NAMESPACE.Name="subscription"
```

\\INSPIRON8200\ROOT:__NAMESPACE.Name="NetFrameworkv1"

> **Tip:** To change the default namespace used in the QueryTool, go to the Management Scope tab.

If you wanted to find all the namespaces within another namespace then you would have to recursively traverse each of the namespaces resetting the default namespace using code. WMI doesn't allow you to perform this type of query natively.

WMI Consumer

Consumers are the top layer in the WMI architecture. A consumer can be a script, enterprise management application, web application, or any other WMI enabled administrative tool. Some consumers can also serve as providers such as Application Center, Operations Manager, and Systems Management Server. In this case they provide information about their resources to other applications while monitoring the resources of other applications via WMI.

Querying with WQL

The Windows Management Instrumentation Query Language (WQL) is a subset of the standard American National Standards Institute Structured Query Language (ANSI SQL) with minor semantic changes to support WMI. WQL supports the following query types:

* **Data queries:** Data queries are used to retrieve class instances and data associations. They are the most commonly used type of query in WMI scripts and applications.
* **Event queries:** Consumers use event queries to register to receive notification of events. Event providers use event queries to register support for one or more events.
* **Schema queries:** Schema queries are used to retrieve class definitions (rather than class instances) and schema associations. Class providers use schema queries to specify the classes that they support when they register.

To understand the various query types I will spend some time on the query syntax so the remaining sections will be clearer.

WQL 101

Like SQL for databases WQL is designed to allow easy retrieval of data from the CIM repository. The core statements which are supported by WQL are:

* **SELECT:** Used by data and event query types, to retrieve class instances. Its purpose is similar to that of SQL. You can also specify the field or property names as in SQL. Also like SQL the more verbose syntax is also the quickest as you only bring back the required data.


```
SELECT * FROM Class                ' Simplier but slower
SELECT Property1, Property2 FROM Class ' More verbose but faster
```

* **ASSOCIATORS OF:** Used by schema query types. There is no SQL equivalent and it is used to return associated class endpoints. More detail on this statement is provided later in the chapter.

```
ASSOCIATORS OF {ClassInstance}
```

* **REFERENCES OF:** Used by schema query types. There is no SQL equivalent. The REFERENCES OF statement is similar to the ASSOCIATORS OF statement in its syntax. However, rather than retrieving endpoint instances, it retrieves the intervening association instances. More detail on this statement is provided later in the chapter.

```
REFERENCES OF {ClassInstance}
```

* **WHERE:** Used by all query types to add criteria in a similar manner to that used in SQL.

```
SELECT * FROM Class WHERE __Class = "Win32_LogicalDisk"
ASSOCIATORS OF {ClassInstance} WHERE SchemaOnly
REFERENCES OF {ClassInstance} WHERE ResultClass = ClassName
```

* **ISA:** Used by all query types to further refine the WHERE clause, it restricts the query to instances that are members of any class deriving from the specified class.

```
SELECT * FROM Class WHERE __this ISA "SubClass"
```

The following table lists the rest of the WQL syntax broken up by those keywords which are related to SQL and those which are not. Where a keyword is prefixed with two underscores (__), they are system related.

Table 2-2. SQL equivalent keywords

WQL keyword	Meaning
AND	Combines two Boolean expressions and returns TRUE when both of the expressions are TRUE.
FALSE	Boolean operator that evaluates to 0.
FROM	Specifies the class that contains the properties listed in a SELECT statement. Unlike SQL, WMI only supports data queries from one class at a time.
IS	Comparison operator used with NOT and NULL. The syntax for this statement is IS [NOT] NULL where NOT is optional.
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. For example: SELECT * FROM meta_class WHERE NOT __class < "Win32" AND NOT __this ISA "Win32_Account"
NULL	Indicates that an object has no explicitly assigned value. NULL is not equivalent

	to zero or blank.
OR	Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators.
TRUE	Boolean operator that evaluates to -1.

Table 2-3. WQL specific keywords

WQL keyword	Meaning
__CLASS	References the class of the object in the query. This keyword is only available in Windows 2000 and later.
__NAMESPACE	Used to query namespaces based on the location defined by the default namespace.
__Win32Provider	Used to query Win32Providers based on the location defined by the default namespace
__TargetInstance	Used in the where clause this keyword specifies that only objects of the intended class or its subclasses are of interest. Used in conjunction with ISA … WHERE __TargetInstance ISA 'Win32_NTLogEvent'
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.
HAVING	Filters the events that are received during the grouping interval specified in the WITHIN clause.
GROUP	Causes WMI to generate a single notification to represent a group of events. Use this clause with event queries.
WITHIN	Specifies either a polling interval or a grouping interval. Use this clause with event queries.

There are many more system related classes contained within the WMI SDK, I recommend that you have a look through the documentation and run some test queries using the query tool against the other classes. You can find a reference to the SDK in the references section of this Chapter.

Examples

It might be best if I show some examples of how these work. The following are examples of what you may wish to do with WMI and WQL.

To make things a little easier you will build a simple query tool, which will allow you to easily run the queries and teach you the basic objects to use at the same time.

There is a bit of UI work in this Query tool, and as I suspect you know basic UI design so I won't go into the detail of how to drop controls on a form as it doesn't really add anything to what we are discussing. The only thing I will mention is that I've made extensive use of the anchor feature. Also you may not be aware that unless you set the MaxLength field of the text box to zero you will by default only be able to store 32768 bytes. You can see the finished tool in Figure 2-2. There are two main screens one for querying and the other to set remote computer properties. In Figure 2-2, I have set the remote user and password, although in this actual instance it's not being used as I'm only using the local machine as defined by the '.'.

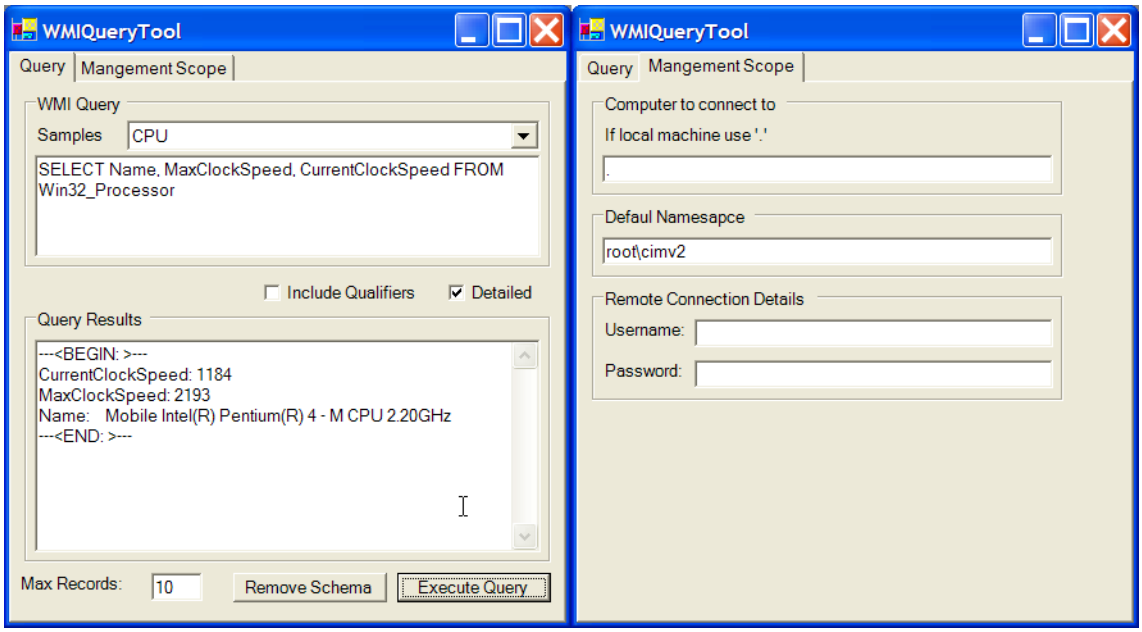

Figure 2-2. WMI Query tool

The core code to get everything working is quite basic; this is because WMI is late bound and so any data query is treated the same way. All the code to run the queries is placed under the Execute Query button:

```
Private Sub btnExecute_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnExecute.Click
    ' This is where all the good stuff is!
    Dim intRecordCount As Integer

    Try ' Errors in your WQL are common so we will watch for that.
        Me.lblStatus.Text = "Status: Processing Query..."
        Application.DoEvents()
        ' Define where to run our WQL Query

        ' Step 1: Define the scope of our WQL Query
        Dim objScope As New ManagementScope(String.Format("\\{0}\{1}", _
            Me.txtMachine.Text, Me.txtDefaultNamespace.Text))
```

```
' Step 2: If a remote machine is selected setup the username & password
If Me.txtMachine.Text <> "." Then
    objScope.Options.Username = Me.txtUserName.Text
    objScope.Options.Password = Me.txtPassword.Text
    objScope.Options.Impersonation = ImpersonationLevel.Impersonate
    objScope.Options.Authentication = AuthenticationLevel.Packet
End If

' Define the options that will be used, including the use of Amended
' Qualifiers
Dim objWMIOptions As New EnumerationOptions()
With objWMIOptions
    .UseAmendedQualifiers = True
End With

' Step 3 & 4: Define the query and supply the scope aswell
Dim objWMISearcher As New ManagementObjectSearcher(objScope, _
    New ObjectQuery(Me.txtQuery.Text), objWMIOptions)
Dim objWMIClass As ManagementBaseObject

' We use the Stingbuilder class here, because well its SUPER! Fast
' compared to simply appending text to the text box.
Dim strbResult As System.Text.StringBuilder = _
    New System.Text.StringBuilder()
Me.txtResults.Text = "" ' Clear any previous results
' Determine what the user wants - Detailed or summary
If Me.chkDetailed.Checked Then
    ' Step 5: Display all the properties for the object (can be long!)
    For Each objWMIClass In objWMISearcher.Get()
        strbResult.Append(String.Format("---<BEGIN: {0}>---{1}", _
            objWMIClass.ToString, vbCrLf))
        Dim objProperty As Management.PropertyData
        For Each objProperty In objWMIClass.Properties
            ' Step 6: Access property details
            ' Some properties can be arrays of varying types, so we pass
            ' it off to a separate function to handle each property
            ' correctly.
            strbResult.Append(objProperty.Name & ": " & _
                Me.GetPropertyData(objProperty))
            ' This code segment adds any Qualifiers to the output too
            If Me.chkQualifiers.Checked Then
                ' Display the qualifiers under the property name, which
                ' can also be of varying Array types too.
                Dim objQualifier As System.Management.QualifierData
                For Each objQualifier In objProperty.Qualifiers
```

```
                    strbResult.Append("     Qualifier: " & _
                        objQualifier.Name & " - " & _
                        Me.GetQualifierData(objQualifier))
                Next
            End If
        Next
        strbResult.Append(String.Format("---<END: {0}>---{1}", _
            objWMIClass.ToString, vbCrLf))
        Me.lblStatus.Text = "Record: " & intRecordCount

        Application.DoEvents()
        ' Check to see if we are only displaying a certain number of
        ' records.
        If CType(Me.txtMaxRecords.Text, Integer) <> 0 Then
            If intRecordCount >= CType(Me.txtMaxRecords.Text, _
                Integer) Then
                Exit For
            End If
        End If
        intRecordCount += 1
    Next
  Else
    ' Step 5: Simply display a one line summary for each object
    For Each objWMIClass In objWMISearcher.Get()
      ' Step 6: This example doesn't access properties, but it could
      strbResult.Append(String.Format("{0}{1}", _
        objWMIClass.ToString, vbCrLf))
    Next
  End If
  ' Render the results to the Text Box
  Me.txtResults.Text = strbResult.ToString
  Me.lblStatus.Text = "Status: Ready..."
 Catch ex As Exception
  ' Opps we have an error
  Me.lblStatus.Text = String.Format("Error: {0}", ex.Message)
  MessageBox.Show("ERROR: " & ex.Message)
 End Try
End Sub

Private Function GetPropertyData(ByVal objProperty As PropertyData) As String
 Dim strResult As String

 If objProperty.IsArray Then
  ' As there are numerous types of array types such as Uint8, string etc
  ' its easier (if not as efficient) to iterate through the collection
  ' coverting each element to its string representation
```

```
        Dim i As Integer

        Dim strbResult As System.Text.StringBuilder = New _
            System.Text.StringBuilder()
        ' Note by default the value property isn't an array. However, we
        ' know it is because we tested the IsArray property first.

        ' As each array type can be different we have to handle each type
        ' seperately as they won't cast to object. The following code is
        ' repetitive but required.
        Select Case TypeName(objProperty.Value)
            Case "Byte()"
                Dim arrItems As Byte()
                arrItems = CType(objProperty.Value, Byte())
                For i = 0 To arrItems.Length - 1
                    strbResult.Append(arrItems(i).ToString)
                    strbResult.Append(", ")
                Next
            Case "String()"
                Dim arrItems As String()

                arrItems = CType(objProperty.Value, String())
                For i = 0 To arrItems.Length - 1
                    strbResult.Append(arrItems(i).ToString)
                    strbResult.Append(", ")
                Next
            Case "Nothing"
                ' There is no data to deal with
                strbResult.Append(", ") ' It will be removed later.
            Case Else
                strbResult.Append("Unknown Array type: " & _
                    TypeName(objProperty.Value) & vbCrLf & ", ")
        End Select

        strResult = strbResult.ToString
        ' remove the last comma before returning the string
        strResult = strResult.Substring(0, strResult.Length - 2)
    Else
        strResult = SafeValue(objProperty.Value)
    End If
    Return strResult & vbCrLf
End Function

Private Function GetQualifierData(ByVal objQualifier As QualifierData) As String
    Dim strResult As String
```

```
    If TypeOf objQualifier.Value Is Array Then
        ' Code ommited here as its basically the same as GetPropertyData
        ' however it uses the objQualifier object instead.
        ...
    Else
        strResult = SafeValue(objQualifier.Value)
    End If
    Return strResult & vbCrLf
End Function

Private Function SafeValue(ByVal objValue As Object) As String
    ' Some properties contain nulls so we make them ""
    If objValue Is Nothing Then
        Return ""
    Else
        Return objValue.ToString
    End If
End Function
```

Tip: When you create your own code, it's not recommended that you dump it all in one routine like has been done here; especially not in the UI.

Although the code appears to be fairly long it's really only because of the extra frills I've added. The core code to get things working without the frills is fairly small and I'll go through the steps to ensure you understand what's happening. The following steps apply to any data or schema type query; I will cover the steps for an event query later.

1. Define the Management Scope, which is used if you need to connect to a remote computer and/or change the default namespace. This step is optional, however if omitted you will use the default scope which will be your local machine and generally the root\cimv2 namespace.
1. Next if you are connecting to a remote machine you may need to supply credentials in the way of a username and password. You assign these details to the instance of the ManagementScope object.
2. Define a WQL Query.
3. Assign the query to an instance of the ManagementObjectSearcher class. This class executes your query and returns a result set of ManagementBaseObject objects.
4. Iterate through the collection of ManagementBaseObject classes returned. This class is a generic view of the class you're interested in; properties of the concrete class are accessed via the Properties collection.
5. Access the properties of the returned class.

Tip: If you are unfamiliar with the use of String.Format, it allows you to define a string with numbered placeholders between braces {0..n}. You can then supply the values for

the placeholders as parameters to the function. This can make your strings easier to read as you can see the actual string more clearly.

This is all the above code is really doing, I have added a bit of extra code to iterate through the properties and qualifiers so that you can have a detailed option in the query tool; I'll discuss the code dealing with qualifiers in more detail when I talk about qualifiers. When you begin to code your own queries you can follow the steps above and add any extra frills as required. More details on the security aspects can be found in the next Chapter.

Note: If you supply property names in your query which don't include those used by the ToString property of the class, the summary result in the query tool will be blank.

Now I'll show some sample WQL which you'll find conveniently contained in the dropdown list so you won't have to type them in to have a go.

How do I find out who's logged onto a particular computer?

This information is contained within the Win32_ComputerSystem class. Therefore to show the username and domain of the currently logged on user you would use:

```
' QueryTool: Current User
SELECT UserName, Domain FROM Win32_ComputerSystem

Result (Detail):
Domain: OFFICE
UserName: INSPIRON\Patricia
```

Now you might be thinking I'm not Patricia (assuming you read the front cover of the book) and you'd right. Using the query tool you just developed I defined the Management Scope to be a remote computer within my network in this case \\INSPIRON, this was done to demonstrate that you can query any computer that is WMI capable.

Is my college's computer better than mine?

Well assuming you have the correct permissions for their computer (which may be doubtful) you can find out all kinds of things about their new computer. The components of most interest are probably the CPU, how much memory it has and what model they managed to score. This requires two separate queries because unlike SQL, WQL doesn't allow more than one class in the FROM clause. The two queries you need are as follows:

```
' QueryTool: CPU
SELECT Name, MaxClockSpeed, CurrentClockSpeed FROM Win32_Processor

Result (Detail):
CurrentClockSpeed: 1184
MaxClockSpeed: 2193
```



```
Name:    Mobile Intel(R) Pentium(R) 4 - M CPU 2.20GHz

' QueryTool: Computer Hardware Specs
SELECT UserName, Model, TotalPhysicalMemory FROM Win32_ComputerSystem

Result (Detail):
Model: Inspiron 8200
TotalPhysicalMemory: 536272896
UserName: INSPIRON8200\Garry
```

> **Note:** WQL does not support cross-namespace queries or associations. You cannot query for all instances of a specified class residing in all the namespaces on the target computer.

So now you know the specs of my machine and how easy it is to do the same with any computer with which you have the correct permissions. This can also be used for more serious work as well. Determining the specs of machines around the office when put into a help desk application would aid in diagnosing issues with applications that require powerful machines. You can also write WQL that will tell you what software is on a given machine, which again can be very useful both from an administrative viewpoint, and possibly as a startup check for your applications; as you may have software dependencies such as Microsoft Excel.

> **Warning:** WQL like its cousin SQL can cause a heavy drain on resources if a poorly constructed query is executed. I'd recommend testing any WQL locally before unleashing it onto someone else's machine.

Although the previous queries may seem trivial they do present the potential power that WMI offers. It also demonstrates that WQL is not that different from its SQL cousin. You can think of a class as being similar to a table and the properties being similar to fields within a table. This also applies to the WHERE clause as you'll see in further examples throughout the Chapter. Where it does start to differ is when you start using events and querying the schema which I will cover shortly.

Meta Data Queries

To finish off this section I'll show you how to query for classes that are contained within the CIM. This can be very useful if you want to know what classes are in a given namespace or you want to know which classes in the CIM derive from a particular super class. The following query returns all the classes within the current namespace.

```
' QueryTool: Meta Data: Class List
SELECT * FROM meta_class

Result (Summary):
\\INSPIRON8200\ROOT\cimv2:__SystemClass
\\INSPIRON8200\ROOT\cimv2:__thisNAMESPACE
```



```
\\INSPIRON8200\ROOT\cimv2:__Provider
\\INSPIRON8200\ROOT\cimv2:__Win32Provider
\\INSPIRON8200\ROOT\cimv2:__IndicationRelated
\\INSPIRON8200\ROOT\cimv2:__EventGenerator
\\INSPIRON8200\ROOT\cimv2:__TimerInstruction
\\INSPIRON8200\ROOT\cimv2:__IntervalTimerInstruction
\\INSPIRON8200\ROOT\cimv2:__AbsoluteTimerInstruction
\\INSPIRON8200\ROOT\cimv2:__Event
...
```

If you were instead interested in all the classes that were directly derived from Win32_Account, then you would use the following query.

```
' QueryTool: Meta Data: Find Super Class
SELECT * FROM meta_class WHERE __SUPERCLASS = 'Win32_Account'

Result (Summary):
\\INSPIRON8200\ROOT\cimv2:Win32_UserAccount
\\INSPIRON8200\ROOT\cimv2:Win32_SystemAccount
\\INSPIRON8200\ROOT\cimv2:Win32_Group
```

Understanding these queries gives you a lot of scope for discovering things about the classes that the CIM holds. You can also experiment with the above queries by turning on the detailed and qualifier options which will show you a lot more detail about the classes.

Data Queries

The most common type of query is a data query as you will mostly want to retrieve instances of classes from the CIM. For example assume you wish to retrieve the names of the applications from the Application Event Log that have a message type of ‘error’. Running the following query in our query tool, produces the following results (note: you will get different results on you computer).

```
' QueryTool: EventLog: Application & Error
SELECT * FROM Win32_NTLogEvent WHERE Logfile = 'Application' and Type='error'

Result (Summary):
\\INSPIRON8200\root\cimv2:Win32_NTLogEvent.Logfile="Application",RecordNumber=86
```

.NET Classes

However, suppose you want to code it yourself, I have already covered the steps of doing this previously. However, the WMI object model does provide other classes to reduce the amount of knowledge you need.

System.Object
  System.Management.ManagementQuery
    System.Management.ObjectQuery
      System.Management.WqlObjectQuery
        System.Management.RelatedObjectQuery
        System.Management.RelationshipQuery
        System.Management.SelectQuery

Figure 2-3. Several objects can be used to query WMI

> **Note:** The following samples can be found in the WQL Samples.vb file in the WMI Samples project.

This code performs the same function as the more generic version; however notice the query I use. I am also passing to the ManagementObjectSearcher class a SelectQuery object instead of the ObjectQuery object I used previously. This object is derived from the ObjectQuery base class (see Figure 2-3).

```
Shared Sub IterateApplicationEventLog()
    ' Step 1: Define Scope
    Dim objScope As New ManagementScope("\\.\root\cimv2")
    ' Step 2: Credentials not required in this case – local machine
    ' Step 3: Define Query
    Dim objWMIQuery As New Management.SelectQuery("Win32_NTLogEvent")
    With objWMIQuery
        .Condition = "Logfile = 'Application' and Type='error'"
    End With
    ' Step 4: Execute Query
    Dim objWMISearcher As New ManagementObjectSearcher(objScope, _
      objWMIQuery)
    Dim objWMIClass As ManagementBaseObject

    ' Step 5: Iterate through results
    For Each objWMIClass In objWMISearcher.Get()
        ' Step 6: Access properties
        Debug.WriteLine(objWMIClass.Properties("SourceName").Value.ToString)
    Next
End Sub
```

When coding data queries you need to follow the following basic steps each time which is independent of the type of class you are trying to access.

1. Define the Management Scope, which is used if you need to connect to a remote computer and/or change the default namespace. This step is optional, however if omitted you will use the default scope which will be your local machine and generally the root\cimv2 namespace.
2. Next if you are connecting to a remote machine you may need to supply credentials in the way of a username and password. You assign these details to the instance of the ManagementScope object.

3. Define a WQL Query, by either using the generic WQL language or by using a more specific query class. When using the SelectQuery class you can supply just the class name. You then define the WHERE clause through the Condition property. You can also define which properties are returned using the SelectedProperties property.
4. Assign the query to an instance of the ManagementObjectSearcher class.
5. Iterate through the collection of ManagementBaseObject classes returned.
6. Access the properties of the returned class.

Using the more specific class may make your code easier to read as you need to set the individual properties. However, I suspect the extra work will make you simply put the full WQL query in. These classes are designed to reduce your knowledge of WMI (ie you don't need to know you use a SELECT statement), but they don't really offer us too much in the way of functionality.

As you can see it doesn't take a great deal of code and it's not any more difficult to do the same for any other class. This is due to the object model being late bound as mentioned earlier.

Event Queries

This would be the second most common query type after data queries. This query type has enormous power and can sometimes be the only way to achieve a given task. The beauty of WMI is that it's remotable and that includes its events engine. When you register to receive an event, you must first determine the type of event you want. WMI supports two types of events: intrinsic events and extrinsic events.

Intrinsic events are events that occur in response to a change in the standard WMI data model. Each intrinsic event class represents a specific type of change. Intrinsic events occur when WMI or a provider creates, deletes, or modifies a namespace, class, or class instance. WMI creates intrinsic events for objects stored within the CIM repository. In contrast, a provider generates intrinsic events for dynamic classes; although WMI can create an instance for a dynamic class if no provider is available. If the provider is unable to provide its own intrinsic event then, WMI uses polling to detect the change which incurs a higher overhead. The following table describes the system classes that WMI uses to report intrinsic events.

Table 2-4 System Event Classes

System class	Description
__ClassCreationEvent	Fires when a class is created.
__ClassDeletionEvent	Fires when a class is deleted.
__ClassModificationEvent	Fires when a class is modified.
__InstanceCreationEvent	Fires when a class instance is created.
__InstanceDeletionEvent	Fires when an instance is deleted.

__InstanceModificationEvent	Fires when an instance is modified.
__NamespaceCreationEvent	Fires when a namespace is created.
__NamespaceDeletionEvent	Fires when a namespace is deleted.
__NamespaceModificationEvent	Fires when a namespace is modified.

An extrinsic event is a predefined occurrence that cannot be directly linked to changes in the WMI data model. Therefore, WMI enables an event provider to define an event class that best describes the event. An example of an extrinsic event would be one that describes a computer switching to stand-by mode. A provider derives an extrinsic event from the __ExtrinsicEvent system class, which is a subclass of the __Event system class. In the next Chapter I will demonstrate how you can add your own extrinsic events to your own classes.

.NET Classes

Taking this information into account, say you wanted to know if a new Event Log entry was created. As you are concerned with the creation event of the Event Log, you would want to attach to the __InstanceCreationEvent of the Win32_NTLogEvent class. The following code shows how you would wire this up using the .NET frameworks classes:

```
Public Class WQL_Samples
    Shared Sub EventWatcher()
        ' Step 1 & 2: Have been ommitted for clarity, see previous examples
        ' Step 3 & 4: Define the Event Query
        Dim objEventWatcher As New ManagementEventWatcher _
          (New WqlEventQuery("SELECT * FROM __InstanceCreationEvent WHERE " & _
          "TargetInstance ISA 'Win32_NTLogEvent'"))

        ' Step 5: Define an instance (although shared methods can be used also)
        '       of the class that has the method to be called when event fires
        Dim objEventArrivedMethod As New WQL_Samples()
        ' Step 5: Attach the method from the instance to EventArrived
        AddHandler objEventWatcher.EventArrived, _
          AddressOf objEventArrivedMethod.Updated

        ' Step 6:Start watching for events
        objEventWatcher.Start()

        ' Just slow things down a bit and wait a bit.
        System.Threading.Thread.Sleep(2000)

        ' Create an Event Log Entry
        Dim objEventLog As New System.Diagnostics.EventLog("Application")
        objEventLog.Source = "WQL Sample"
        objEventLog.WriteEntry("Sample WQL Entry...")
```

```
        ' Wait for event to fire.
        System.Threading.Thread.Sleep(2000)
        ' Step 8: Stop waiting for events
        objEventWatcher.Stop() ' Stops waiting for events.
    End Sub

    ' This is the method that will be called when the event fires.
    Sub Updated(ByVal sender As Object, ByVal e As EventArrivedEventArgs)
        Dim objClassInstance As ManagementBaseObject

        ' Step 7: Access the details from the event
        objClassInstance = CType(e.NewEvent. _
            Properties("TargetInstance").Value, ManagementBaseObject)
        MessageBox.Show("The following entry was just added: " & _
            objClassInstance.Properties("LogFile").Value.ToString & ": " & _
            objClassInstance.Properties("Message").Value.ToString)
    End Sub
End Class
```

System.Object
  System.Management.ManagementQuery
    System.Management.EventQuery
      System.Management.WqlEventQuery

Figure 2-4. Event Query hierarchy

Figure 2-4 shows the hierarchy for event queries. When coding event queries you need to follow these basic steps each time which are independent of the type of class you are trying to access.

2. . Define the Management Scope, which is used if you need to connect to a remote computer and/or change the default namespace. This step is optional, however if omitted you will use the default scope which will be your local machine and generally the root\cimv2 namespace.
7. . Next if you are connecting to a remote machine you may need to supply credentials in the way of a username and password. You assign these details to the instance of the ManagementScope object.
8. . Define a WQL Query, which includes the event and the class. Note you can supply a full WQL query or use the many overloaded methods to supply the query details.
9. . Assign the query to an instance of the ManagementEventWatcher class.
10. . Attach the method to be called once the event fires to the EventArrived method of the ManagementEventWatcher class instance. This is done by passing the address of the method to be called using the AddressOf operator.

11. . Wait for the event to fire by calling the Start method of the ManagementEventWatcher class instance.
12. . Access the properties of the returned class via the EventArrivedEventArgs when the event fires (assuming it ever does), in the method that handles the event.
13. . Once you no longer want to receive notifications of events then you call the Stop method of the ManagementEventWatcher class instance.

Although there is more code here it's not particularly difficult. The only area of difficulty is where you hook up the method to be called when the event fires. .NET is making use of delegates here and if you're not too familiar with them search for 'delegates' in the VS.NET online help for more information.

Qualifiers

Qualifiers are additional information added to the schema to describe useful information about the classes and its methods, parameters and properties. Each qualifier is a name valued pair which can have varying data types such as string, integer, byte or Boolean. A very common and useful qualifier is Key which tells us if a particular property of a class is the key; its value if present is True or False. In this example, the qualifier would normally only be present if the value were True. This means you can't expect a particular qualifier to be always present; you will need to test for its existence and then retrieve its value.

Sometimes a particular qualifier will not be a basic type such as a string or Boolean. In some circumstances, it will be an array of basic types. In many cases there may be two arrays qualifiers, one that specifies a system enumeration such as 1, 2, 3, 4, 5, 6... 20. The other specifying a localized description of the enumeration such as FILE_READ_DATA (file) or FILE_LIST_DIRECTORY (directory), FILE_WRITE_DATA (file) or FILE_ADD_FILE (directory), FILE_APPEND_DATA (file) or FILE_ADD_SUBDIRECTORY (directory). You can see how this example works by specifying in the query tool to display both a detailed list and their qualifiers as shown in Figure 2-5.

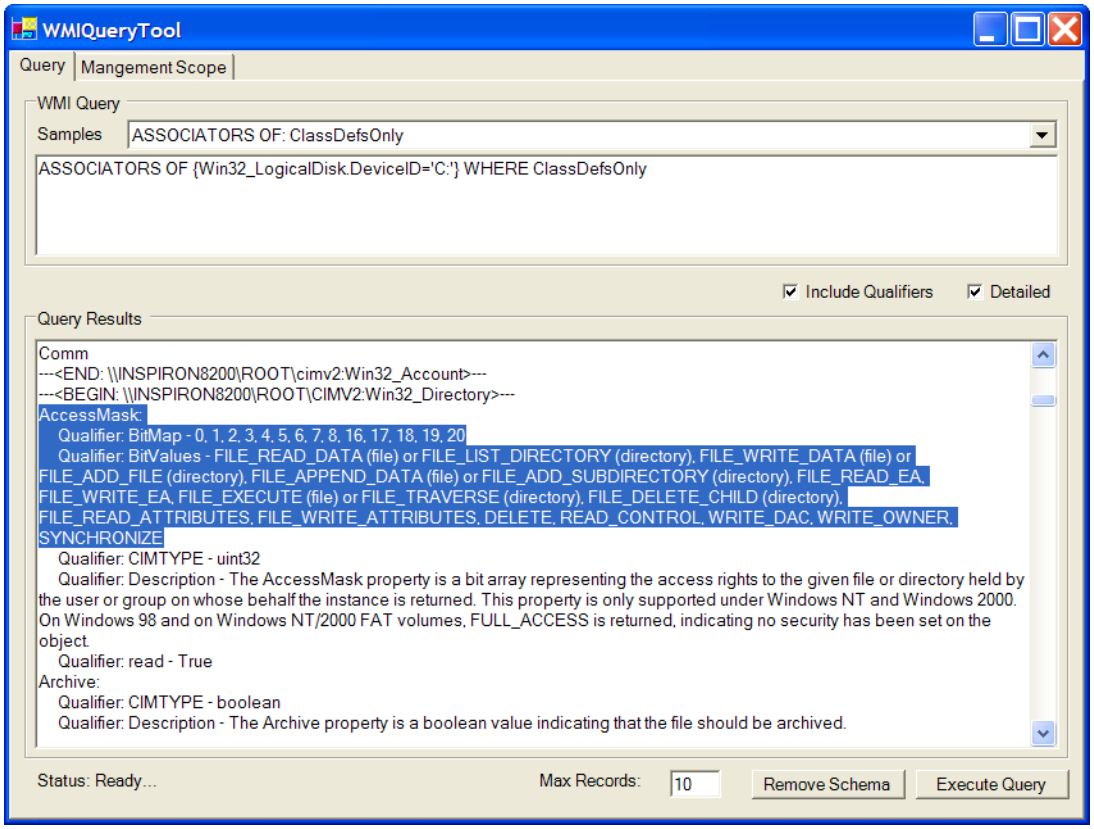

Figure 2-5. Query Tool Class Qualifiers

You will also notice that qualifiers describe other aspects of the property too, such as the properties data type and description. The type of qualifiers and the number of them will vary greatly between class types. Also it should be noted that many of the qualifiers are not available when you have an instance of the class. The qualifiers can be dependant on whether you are looking at the class description or its instance. Taking a look forward to Figure 2-8 you can see that instance classes show qualifiers that are propagated to the instance only.

Accessing Qualifier Data

Earlier when we created the query tool, I mentioned I'd cover how the code for accessing qualifier data worked.

```
Private Function GetQualifierData(ByVal objQualifier As QualifierData) As String
    Dim strResult As String

    If TypeOf objQualifier.Value Is Array Then
        ' As there are numerous types of array types such as Uint8, string etc
        ' its easier (if not as efficient) to iterate through the collection
        ' coverting each element to its string representation
        Dim i As Integer
```

```
        Dim strbResult As System.Text.StringBuilder = _
            New System.Text.StringBuilder()
        ' Note by default the value property isn't an array. However, we know it
        ' is because we tested to see if the Value was of type Array.

        ' As each array type can be different we have to handle each type
        ' seperately as they won't cast to object. The following code is
        ' repetitive but required.
        Select Case TypeName(objQualifier.Value)
            Case "Byte()"
                Dim arrItems As Byte()
                arrItems = CType(objQualifier.Value, Byte())
                For i = 0 To arrItems.Length - 1
                    strbResult.Append(arrItems(i).ToString)
                    strbResult.Append(", ")
                Next
            Case "String()"
                Dim arrItems As String()
                arrItems = CType(objQualifier.Value, String())
                For i = 0 To arrItems.Length - 1
                    strbResult.Append(arrItems(i).ToString)
                    strbResult.Append(", ")
                Next
            Case "Nothing"
                ' There is no data to deal with
                strbResult.Append(", ") ' It will be removed later.
            Case Else
                strbResult.Append("Unknown Array type: " & _
                    TypeName(objQualifier.Value) & vbCrLf & ", ")
        End Select

        strResult = strbResult.ToString
        ' remove the last comma before returning the string
        strResult = strResult.Substring(0, strResult.Length - 2)
    Else
        strResult = SafeValue(objQualifier.Value)
    End If
    Return strResult & vbCrLf
End Function
```

The previous code is very similar to that of the GetPropertyData function. However, because the two classes we want to examine don't inherit from a common class we have to create two copies of virtually the same thing. You'll note that checking if the value property is an array is different in each routine as the QualifierData object doesn't support the IsArray property like the PropertyData object does.

In essence the code is designed to convert an array value type into a simple comma separated string for easy viewing. Because the array can be of varying types we have to code for each array type individually. I've made use of the string builder class as some arrays can be fairly large and standard string concatenation in .Net is not very efficient. Also as VB.NET doesn't support unsigned types which are commonly used in WMI, each value is converted to its string counterpart, which allows VB.NET to handle the value.

CIM Studio

Although the query tool we have built is good for running ad hoc queries, it can be a little awkward to use for researching what is in the CIM. As part of the WMI SDK you get a few tools one of which is CIM Studio. This web based utility is excellent for browsing and searching the CIM meta data.

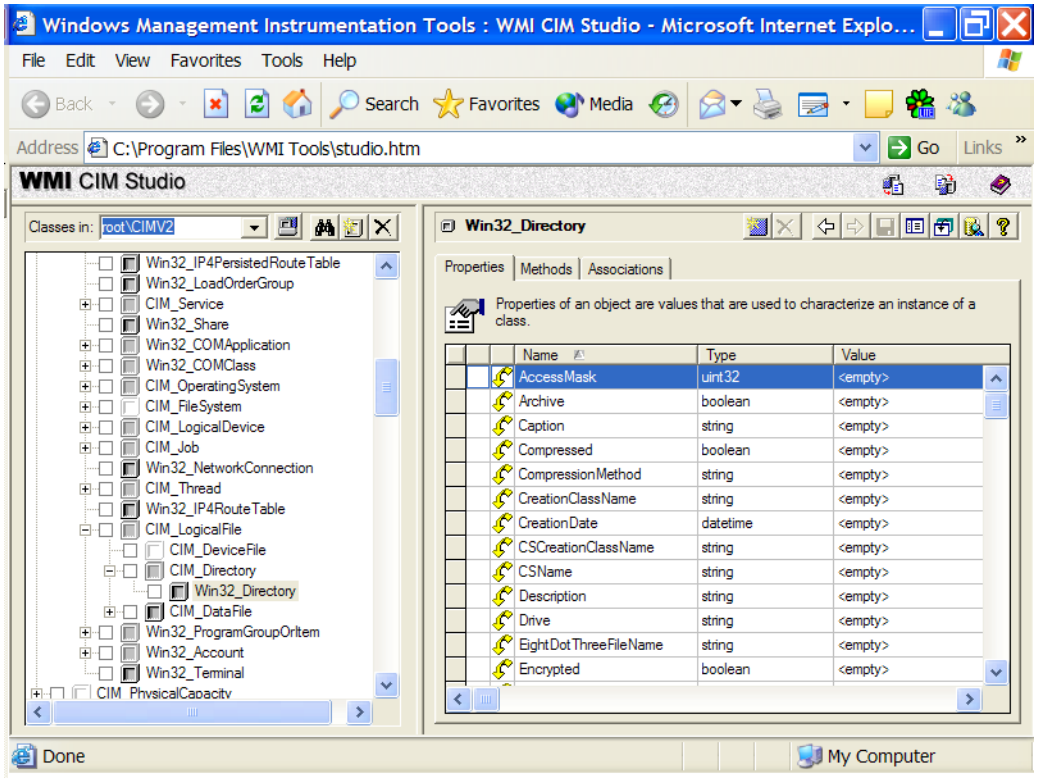

Figure 2-6. Query Tool Class Qualifiers

Figure 2-6 shows you what CIM Studio looks like where you can see that CIM Studio displays all the classes in the left pane and class details in the right pane. You can access all the properties, methods and associations of any class by switching between the views in the right pane. If you double click on the icon to the left of the property name it will bring up the qualifiers for that property too; an example is given in Figure 2-8.

A detailed discussion of how to use CIM Studio is outside the scope of this book, however it is important that you know that it exists. There is a help file which comes with the program which I suggest you have a look through to get the best out of the product. I

would suggest that this will be a very popular tool for those of you who start to use WMI regularly.

Associations

As you'll remember each piece of information that is stored in the CIM is exposed as classes which are templates for class instances or objects. As with any object model, objects don't normally live in isolation, rather they are related to each other in some way. For example each disk drive you have in your computer is related to many other parts of the computer. For example the logical disk drive C:\ is related to parts of the operating system such as disk quotas, partitions and directories. These are then related to other parts of the system such as users through the security settings that they have. As you can see almost every class within the CIM is related to another class in some way.

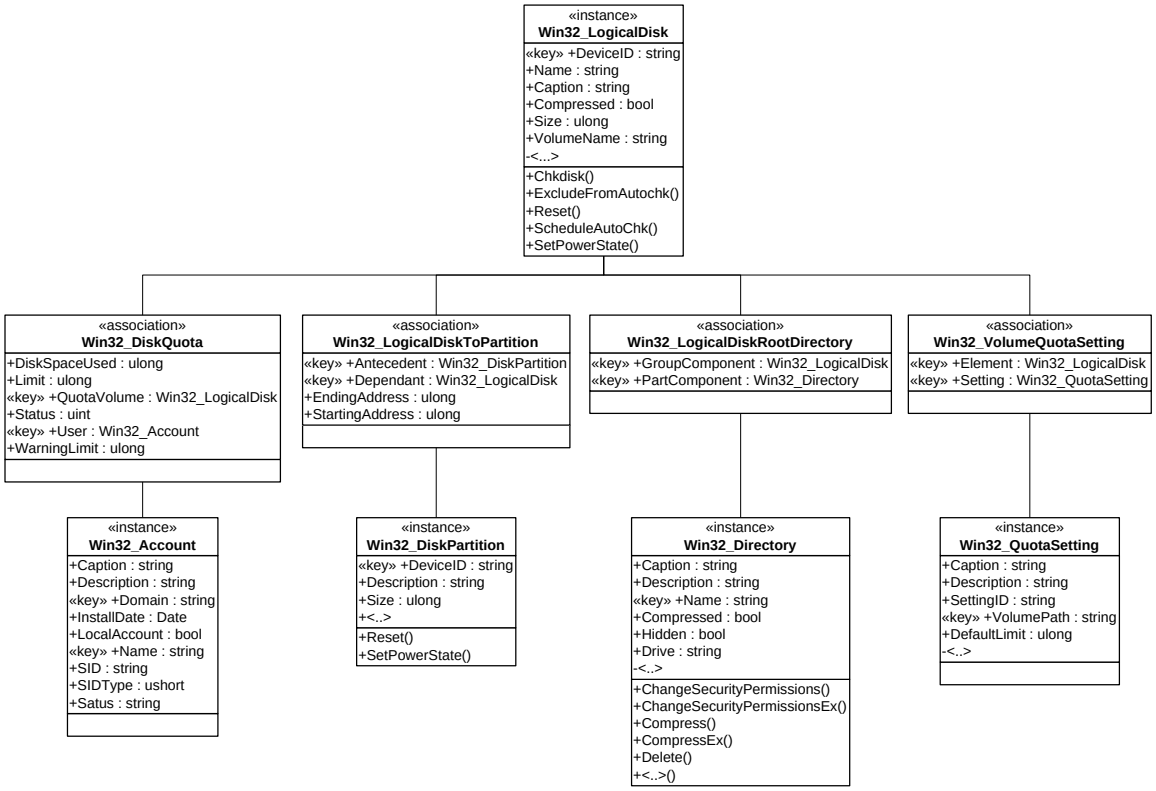

Figure 2-7. Sample Associations

WMI allows us to explore these relationships through associations which are the intermediate classes that join other classes together. For example if you wanted to split a logical disk into several partitions you would need three classes. An instance class which describes the logical disk (LogicalDisk), an instance class to describe the partition (DiskPartition) and an association class which related them together (LogicalDiskToPartition). You can use CIM Studio to view these relationships, but selecting the class you're interested in, then selecting the associations tab in the right pane.

Using CIM Studio I was able to diagram the relationships for the Win32_LogicalDisk class which is presented in Figure 2-7. The diagram clearly shows the association classes that are related to the Win32_LogicalDisk class.

> **Note:** There were many properties on several classes. The use of an ellipse (<…>) indicates that some were removed to keep the class description brief.

WMI uses schema queries to allow us to explore these relationships using two keywords ASSOCIATORS OF and REFERENCES OF; the details of each are explained next.

ASSOCIATORS OF Statement

The ASSOCIATORS OF statement is used to retrieve the associated classes of a given class instance not its associating classes. This means that for our class diagram the associated classes are Win32_Account, Win32_DiskPartition, Win32_Directory and Win32_QuotaSetting as shown on Figure 2-7.

The full syntax of the ASSOCIATORS OF statement is given below. Note that all sections of the WHERE clause are optional and don't rely on each other. The only part that is mandatory is the first section which specifies the class instance. Take note to include the braces ({…}) as they are part of the required syntax:

```
ASSOCIATORS OF {ClassInstance} WHERE
  AssocClass = AssocClassName
  ClassDefsOnly
  RequiredAssocQualifier = QualifierName
  RequiredQualifier = QualifierName
  ResultClass = ClassName
  ResultRole = PropertyName
  Role = PropertyName
```

ClassDefsOnly

Adding ClassDefsOnly to the WHERE class means that the result will only be the class definitions not the instances themselves. This allows the result to return much faster as we are not returning any instance data just the class definitions. To illustrate we will return the classes associated with the Win32_LogicalDisk as represented in our class diagram:

```
QueryTool: ASSOCIATORS OF: ClassDefsOnly
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID="C:"} WHERE ClassDefsOnly

Results (Summary):
\\INSPIRON8200\ROOT\cimv2:Win32_Account
\\INSPIRON8200\ROOT\CIMV2:Win32_Directory
\\INSPIRON8200\ROOT\CIMV2:Win32_ComputerSystem
\\INSPIRON8200\ROOT\CIMV2:Win32_QuotaSetting
```


\\INSPIRON8200\ROOT\CIMV2:Win32_DiskPartition

As you can see the result is the same as bottom level of our class diagram meaning that the ASSOCIATORS OF keyword has returned the instance classes not the association classes which can be a little confusing. You may notice that there's an extra class Win32_ComputerSystem this class is at a higher level, but it is still associated with a logical disk. I didn't want to model from that high up as there's so many associated classes. The following WQL shows what I mean; note you'll need to change inspiron8200 to the name of your computer to run this on your system:

```
QueryTool: ASSOCIATORS OF: ComputerSystem – ClassDefsOnly
ASSOCIATORS OF {Win32_ComputerSystem.Name='inspiron8200'} WHERE ClassDefsOnly

Results (Summary):
\\INSPIRON8200\ROOT\cimv2:Win32_BIOS
\\INSPIRON8200\ROOT\cimv2:Win32_Process
\\INSPIRON8200\ROOT\cimv2:Win32_LoadOrderGroup
\\INSPIRON8200\ROOT\cimv2:Win32_LogicalProgramGroup
\\INSPIRON8200\ROOT\cimv2:Win32_Desktop
\\INSPIRON8200\ROOT\cimv2:Win32_BootConfiguration
\\INSPIRON8200\ROOT\cimv2:Win32_SystemDriver
\\INSPIRON8200\ROOT\cimv2:Win32_UserAccount
\\INSPIRON8200\ROOT\cimv2:Win32_TimeZone
\\INSPIRON8200\ROOT\cimv2:Win32_OperatingSystem
\\INSPIRON8200\ROOT\cimv2:Win32_LogicalMemoryConfiguration
\\INSPIRON8200\ROOT\cimv2:Win32_Service
\\INSPIRON8200\ROOT\cimv2:Win32_NTLogEvent
\\INSPIRON8200\ROOT\cimv2:Win32_Environment
\\INSPIRON8200\ROOT\cimv2:Win32_DMAChannel
\\INSPIRON8200\ROOT\cimv2:Win32_IRQResource
\\INSPIRON8200\ROOT\cimv2:Win32_DeviceMemoryAddress
\\INSPIRON8200\ROOT\cimv2:Win32_PortResource
\\INSPIRON8200\ROOT\cimv2:Win32_SoftwareElement
\\INSPIRON8200\ROOT\cimv2:Win32_Bus
\\INSPIRON8200\ROOT\cimv2:Win32_MotherboardDevice
\\INSPIRON8200\ROOT\cimv2:Win32_PnPEntity
\\INSPIRON8200\ROOT\cimv2:Win32_SoundDevice
\\INSPIRON8200\ROOT\cimv2:Win32_1394Controller
\\INSPIRON8200\ROOT\cimv2:Win32_Battery
\\INSPIRON8200\ROOT\cimv2:Win32_FloppyController
\\INSPIRON8200\ROOT\cimv2:Win32_IDEController
\\INSPIRON8200\ROOT\cimv2:Win32_NetworkAdapter
\\INSPIRON8200\ROOT\cimv2:Win32_PortableBattery
\\INSPIRON8200\ROOT\cimv2:Win32_POTSModem
\\INSPIRON8200\ROOT\cimv2:Win32_Printer
\\INSPIRON8200\ROOT\cimv2:Win32_Processor
\\INSPIRON8200\ROOT\cimv2:Win32_CDROMDrive
```



```
\\INSPIRON8200\ROOT\cimv2:Win32_DiskDrive
\\INSPIRON8200\ROOT\cimv2:Win32_DiskPartition
\\INSPIRON8200\ROOT\cimv2:Win32_Fan
\\INSPIRON8200\ROOT\cimv2:Win32_FloppyDrive
\\INSPIRON8200\ROOT\cimv2:Win32_Keyboard
\\INSPIRON8200\ROOT\cimv2:Win32_LogicalDisk
\\INSPIRON8200\ROOT\cimv2:Win32_MemoryArray
\\INSPIRON8200\ROOT\cimv2:Win32_MemoryDevice
\\INSPIRON8200\ROOT\cimv2:Win32_ParallelPort
\\INSPIRON8200\ROOT\cimv2:Win32_PCMCIAController
\\INSPIRON8200\ROOT\cimv2:Win32_PointingDevice
\\INSPIRON8200\ROOT\cimv2:Win32_SerialPort
\\INSPIRON8200\ROOT\cimv2:Win32_USBController
\\INSPIRON8200\ROOT\cimv2:Win32_USBHub
\\INSPIRON8200\ROOT\cimv2:Win32_CacheMemory
\\INSPIRON8200\ROOT\cimv2:Win32_DesktopMonitor
\\INSPIRON8200\ROOT\cimv2:Win32_TemperatureProbe
\\INSPIRON8200\ROOT\cimv2:Win32_VideoController
```

You'll also notice that the Win32_LogicalDisk was one of the associations.

AssocClass

Using the AssocClass keyword allows us to restrict the result to the class instances that are derived from the specified class. This is useful if you are only concerned with one type of association class such as directory information for the root drive as contained in the Win32_LogicalDiskRootDirectory association class:

```
QueryTool: ASSOCIATORS OF: AssocClass
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID="C:"}
   WHERE AssocClass = Win32_LogicalDiskRootDirectory

Results (Summary):
\\INSPIRON8200\root\cimv2:Win32_Directory.Name="c:\\"
```

RequiredAssocQualifier

This keyword indicates that the returned instances must be associated with the source class through an association class that includes the specified qualifier. This type of filtering is used to eliminate broad ranges of associated classes unless the classes are associated with the target through a particular set of tagged association classes. The following example returns class instances if the association class includes a qualifier called Association:

```
QueryTool: ASSOCIATORS OF: RequiredAssocQualifier
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}
   WHERE RequiredAssocQualifier = Association
```


Results (Summary):

' The following prefix was removed for clarity: \\INSPIRON8200\root\cimv2:

Win32_Directory.Name="c:\\"

Win32_SystemAccount.Domain="INSPIRON8200",Name="Administrators"

Win32_ComputerSystem.Name="INSPIRON8200"

Win32_QuotaSetting.VolumePath="C:\\"

Win32_DiskPartition.DeviceID="Disk #0, Partition #0"

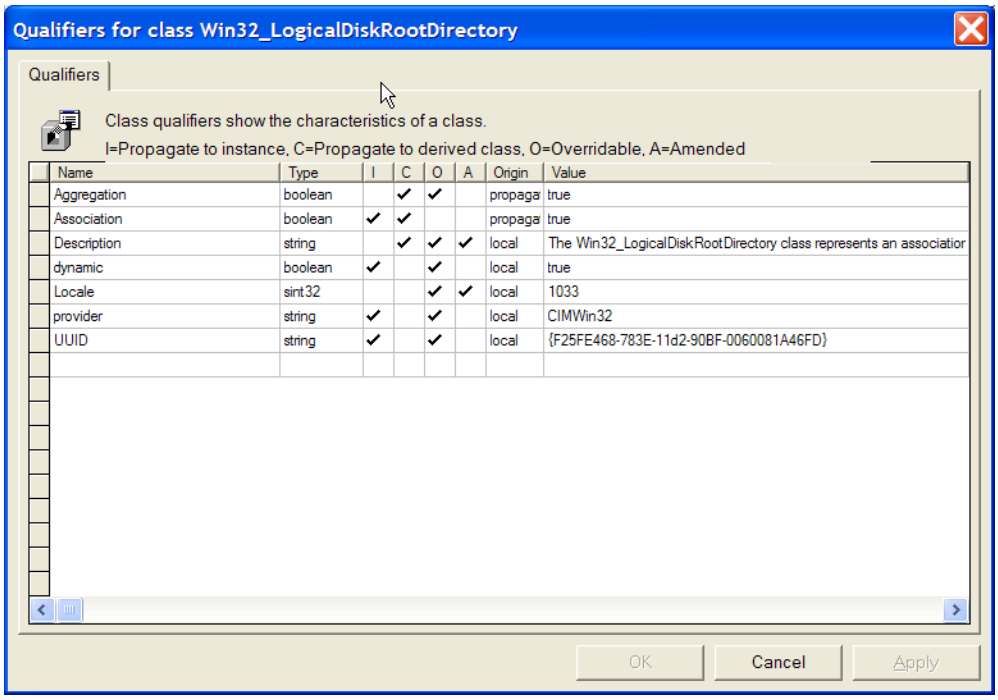

Figure 2-8. Class Qualifiers

You can see what qualifiers are associated with a class by using CIM Studio and selecting Object Qualifiers from the context menu. The result for Win32_LogicalDiskRootDirectory is shown in Figure 2-8 which shows the Association qualifier which was queried for.

RequiredQualifier

This keyword indicates that the returned classes associated with the source class must include the specified qualifier. The RequiredQualifier keyword can be used to include particular types of instances in the result set. The following example returns endpoint instances that include a qualifier called Locale:

QueryTool: ASSOCIATORS OF: **RequiredQualifier**

ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}

WHERE RequiredQualifier = Locale

Results (Summary):

```
' The following prefix was removed for clarity: \\INSPIRON8200\root\cimv2:
Win32_Directory.Name="c:\\"
Win32_SystemAccount.Domain="INSPIRON8200",Name="Administrators"
Win32_ComputerSystem.Name="INSPIRON8200"
Win32_QuotaSetting.VolumePath="C:\\"
Win32_DiskPartition.DeviceID="Disk #0, Partition #0"
```

ResultClass

This keyword indicates that the returned classes associated with the source class must belong to or be derived from the specified class. The following example returns class instances that are derived from the CIM_Directory class:

```
QueryTool: ASSOCIATORS OF: ResultClass
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}
   WHERE ResultClass = Cim_Directory

Results (Summary):
\\INSPIRON8200\root\cimv2:Win32_Directory.Name="c:\\"
```

> Note: The ClassDefsOnly and ResultClass keywords are mutually exclusive and will cause an invalid query error if used together. Also complex queries cannot use "And" or "Or" to separate keywords for ASSOCIATORS OF and REFERENCES OF statements. The equal sign is also the only valid operator that can be used with the keywords in these queries.

You can discover what a given class is derived from easily by using the CIM Studio. Simply click the Array property of the __DERIVATION attribute of the class you're interested in. Figure 2-9 shows the classes that Win32_Directory is derived from.

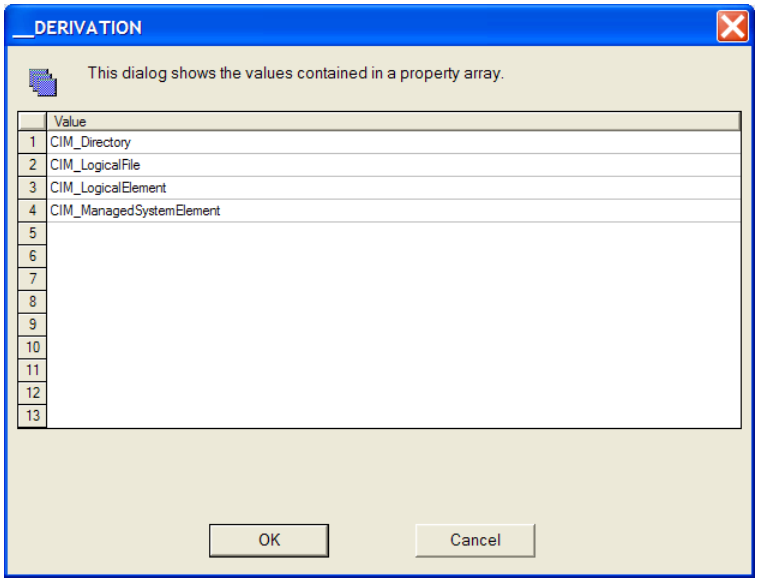

Figure 2-9. Class Derivations

ResultRole

This keyword indicates that the returned classes participate in an association with the source class where the source class plays a particular role. The role is defined by the specified property, a reference property of type ref. For example, the Role keyword can be used to retrieve all endpoints associated with a source object that have the GroupComponent property, as the following query illustrates:

```
QueryTool: ASSOCIATORS OF: ResultRole
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}
   WHERE ResultRole = GroupComponent

Results (Summary):
\\INSPIRON8200\root\cimv2:Win32_ComputerSystem.Name="INSPIRON8200"
```

Role

This keyword indicates that the returned class instances participate in an association with the source class where the source class plays a particular role. The role is defined by the specified property, a reference property of type ref. For example, the Role keyword can be used to retrieve all classes associated with a source class that have the GroupComponent property:

```
QueryTool: ASSOCIATORS OF: Role
ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'} WHERE Role = GroupComponent

Results (Summary):
\\INSPIRON8200\root\cimv2:Win32_Directory.Name="c:\\"
```

.NET Classes

You now know the query syntax but how do you wire that up to the managed classes within the .NET framework. The next example shows how you can reproduce this output:

```
Public Shared Sub AssociatorsOF()
   'This query requests all objects related to the 'C:' drive.
   Dim objWMIQuery As New RelatedObjectQuery("win32_logicaldisk='c:'")
   Dim objWMISearcher As New ManagementObjectSearcher(objWMIQuery)

   Dim objClass As ManagementObject
   ' Now iterate through each of the associations return by the query
   For Each objClass In objWMISearcher.Get()
      Debug.WriteLine(objClass.ToString())
   Next
End Sub
```

Again I am making use of the more specific class RelatedObjectQuery although you could use the more generic ObjectQuery class if you wish. In the example, coding

ASSOCIATORS OF schema queries requires the following basic steps each time which is independent of the type of class you are trying to access.

1. Identify the class instance you are interested being the ObjectPath, our example makes use of the RelatedQueryObject for querying which builds the correct WQL. If however if you wish to use the full WQL syntax you can use either ObjectQuery or RelatedQueryObject.
2. Assign the query to an instance of the ManagementObjectSearcher class.
3. Iterate through the collection of ManagementBaseObject classes returned.
4. Access the properties of the returned class.

REFERENCES OF Statement

Now that you have an understanding of the ASSOCIATORS OF statement it's fairly easy to explain the use of the REFERENCES OF statement. The main difference is that instead of returning the class instances that are associated with a given class instance, it returns the association classes instead.

The full syntax of the REFERENCES OF statement is given below. Note that all sections of the WHERE clause are optional and don't rely on each other. The only part that is mandatory is the first section which specifies the object path. Take note to include the braces ({…}) as they are part of the required syntax:

```
REFERENCES OF {ClassInstance} WHERE
    ClassDefsOnly
    RequiredQualifier = QualifierName
    ResultClass = ClassName
    Role = PropertyName
```

The keywords used for the REFERENCES OF statement match the meaning of their ASSOCIATORS OF counterpart. The results however are obviously different given the beginning statements are different. The following will show the results using the REFERENCES OF statement. You can compare the results to those of the ASSOCIATORS OF statement presented earlier.

ClassDefsOnly

Adding ClassDefsOnly to the WHERE class like the ASSOCIATORS OF statement means that the result will only be the class definitions not the instances themselves. This allows the result to return much faster as we are not returning any instance data just the class definitions. To illustrate we will return the classes associated with the Win32_LogicalDisk as represented in our class diagram:

```
QueryTool: REFERENCES OF: ClassDefsOnly
REFERENCES OF {Win32_LogicalDisk.DeviceID="C:"} WHERE ClassDefsOnly

Results (Summary):
```



```
\\INSPIRON8200\ROOT\cimv2:Win32_LogicalDiskRootDirectory
\\INSPIRON8200\ROOT\cimv2:Win32_DiskQuota
\\INSPIRON8200\ROOT\cimv2:Win32_SystemDevices
\\INSPIRON8200\ROOT\cimv2:Win32_VolumeQuotaSetting
\\INSPIRON8200\ROOT\cimv2:Win32_LogicalDiskToPartition
```

Notice now that the result is the association classes that are used to associate the Win32_LogicalDisk with the associated classes such as Win32_Account, Win32_DiskPartition, Win32_Directory and Win32_QuotaSetting. You can see how they relate to the Win32_LogicalDisk by looking back at Figure 2-7.

RequiredQualifier

This keyword indicates that the returned classes associated with the source class must include the specified qualifier. The RequiredQualifier keyword can be used to include particular types of instances in the result set. The following example returns endpoint instances that include a qualifier called Locale:

```
QueryTool: REFERENCES OF: RequiredQualifier
REFERENCES OF {Win32_LogicalDisk.DeviceID='C:'}
   WHERE RequiredQualifier = Locale

Results (Summary):
' The following prefix was removed for clarity: \\INSPIRON8200\root\cimv2:
Win32_DiskQuota.QuotaVolume="Win32_LogicalDisk.DeviceID=\"C:\"",User="Win32_Account.Dom
ain=\"INSPIRON8200\",Name=\"Administrators\""

Win32_LogicalDiskRootDirectory.GroupComponent="\\\\INSPIRON8200\\root\\CIMV2:Win32_Logica
lDisk.DeviceID=\"C:\"",PartComponent="\\\\INSPIRON8200\\root\\cimv2:Win32_Directory.Name=\"C:
\\\\\""

Win32_SystemDevices.GroupComponent="\\\\INSPIRON8200\\root\\cimv2:Win32_ComputerSystem
.Name=\"INSPIRON8200\"",PartComponent="\\\\INSPIRON8200\\root\\cimv2:Win32_LogicalDisk.De
viceID=\"C:\""

Win32_VolumeQuotaSetting.Element="\\\\INSPIRON8200\\root\\cimv2:Win32_LogicalDisk.DeviceID
=\"C:\"",Setting="\\\\INSPIRON8200\\root\\cimv2:Win32_QuotaSetting.VolumePath=\"C:\\\\\""

Win32_LogicalDiskToPartition.Antecedent="\\\\INSPIRON8200\\root\\cimv2:Win32_DiskPartition.De
viceID=\"Disk #0, Partition
#0\"",Dependent="\\\\INSPIRON8200\\root\\cimv2:Win32_LogicalDisk.DeviceID=\"C:\""
```


ResultClass

This keyword indicates that the returned classes associated with the source class must belong to or be derived from the specified class. The following example returns class instances that are derived from the Win32_LogicalDiskToPartition class:

```
QueryTool: REFERENCES OF: ResultClass
REFERENCES OF {Win32_LogicalDisk.DeviceID='C:'}
   WHERE ResultClass = Win32_LogicalDiskToPartition

Results (Summary):
\\INSPIRON8200\root\cimv2:Win32_LogicalDiskToPartition.Antecedent="\\\\INSPIRON8200\\root\\ci
mv2:Win32_DiskPartition.DeviceID=\"Disk #0, Partition
#0\"",Dependent="\\\\INSPIRON8200\\root\\cimv2:Win32_LogicalDisk.DeviceID=\"C:\""
```

Role

This keyword indicates that the returned class instances participate in an association with the source class where the source class plays a particular role. The role is defined by the specified property, a reference property of type ref. For example, the Role keyword can be used to retrieve all classes associated with a source class that have the GroupComponent property:

```
QueryTool: REFERENCES OF: Role
REFERENCES OF {Win32_LogicalDisk.DeviceID='C:'} WHERE Role = GroupComponent

Results (Summary):
Win32_LogicalDiskRootDirectory.GroupComponent="\\\\INSPIRON8200\\root\\CIMV2:Win32_Logica
lDisk.DeviceID=\"C:\"",PartComponent="\\\\INSPIRON8200\\root\\cimv2:Win32_Directory.Name=\"C:
\\\\\""
```

Summary

At the beginning of the Chapter I mentioned that WMI was Microsoft's best kept secret. I trust that having read this Chapter you can see why. WMI is a very powerful technology, which if you let yourself think about its possibilities, might find its way into your next enterprise development project.

We learned that there is an enormous amount of information that WMI has; which is easily accessible using WQL, an SQL derivative. We also learned that we can use WMI across the enterprise due to its remotable capability. It was also discovered that subscribing to events allows us to be notified when an object is created, deleted or modified.

The WMI story however isn't over and the next Chapter will build on the knowledge we have just learned. In the next Chapter we will look at the ability to create our own WMI providers, allowing you to expose both an objects data and its events in only a few

lines of code. If you haven't been convinced by the power of WMI yet, you will be by the end of the next Chapter.

Resources:

WMI Managed Extensions for VS.NET 2002:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/WMISE/wmioriManagingWindowsManagementInstrumentationWMIDataEventsWithServerExplorer.asp

WMI Managed Extensions for VS.NET 2003:

http://www.microsoft.com/downloads/details.aspx?FamilyID=62d91a63-1253-4ea6-8599-68fb3ef77de1&DisplayLang=en

COM API for WMI:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/com_api_for_wmi.asp

WMI SDK (a must have download!)

http://msdn.microsoft.com/downloads/sdks/wmi/default.asp