

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

Since Windows NT's inception, the OS has incorporated integrated performance and system-event monitoring tools. Applications and the system typically use the Event Manager to report errors and diagnostic messages. The Event Viewer utility, which Microsoft built into NT, lets administrators view event output from either the local computer or another computer on the network. Similarly, the Performance API lets applications and OS subsystems report performance-related statistics that administrators then use the Performance Monitor utility to view.

Although NT's event- and performance-monitoring features meet their design goals, these features have limitations. For example, the facilities' programming interfaces differ from one another, and this variation increases the complexity of applications that use both event and performance monitoring to collect data. The Performance API's performance can be poor, especially across a network, because the API is an all-or-nothing proposition: No way exists for an application to query the performance information of only specific components. Perhaps the biggest drawback to the existing monitoring facilities is that they have little or no extensibility, and neither provides the two-way interaction necessary in a management API. Applications must provide data in predefined formats for event logging or performance-data collection. The Performance API provides no way for an application to receive notification of performance-related events, and applications that request notification of Event Manager events can't restrict notification to specific event types or sources. Finally, clients of either collection facility can't communicate with event- or performance-data providers through the Event Manager or Performance API.

Windows Management Interface (WMI) is Microsoft's next step in Windows enterprise-management support. WMI is Microsoft's implementation of Web-Based Enterprise Management (WBEM) technology. WBEM is a standard that the Distributed Management Task Force (DMTF—an industry consortium) defines. The WBEM standard encompasses the design of an extensible enterprise data-collection and management facility that has the flexibility and extensibility required to manage local and remote systems that comprise arbitrary components. Microsoft implemented WMI on Windows 98, made WMI available for NT 4.0 with Service Pack 4 (SP4) or later and Win95 OSR2, and integrated WMI into Windows 2000 (Win2K). However, Microsoft's WMI development targets Win2K. As third parties begin to implement WMI-based management tools, administrators will gain the ability to monitor and control all aspects of their Win2K, NT, or Win9x systems from anywhere in their enterprise. In this column, I take you inside WMI. Most of what I describe applies to all the Windows platforms that support WMI, but when I discuss implementation details, I focus on Win2K.

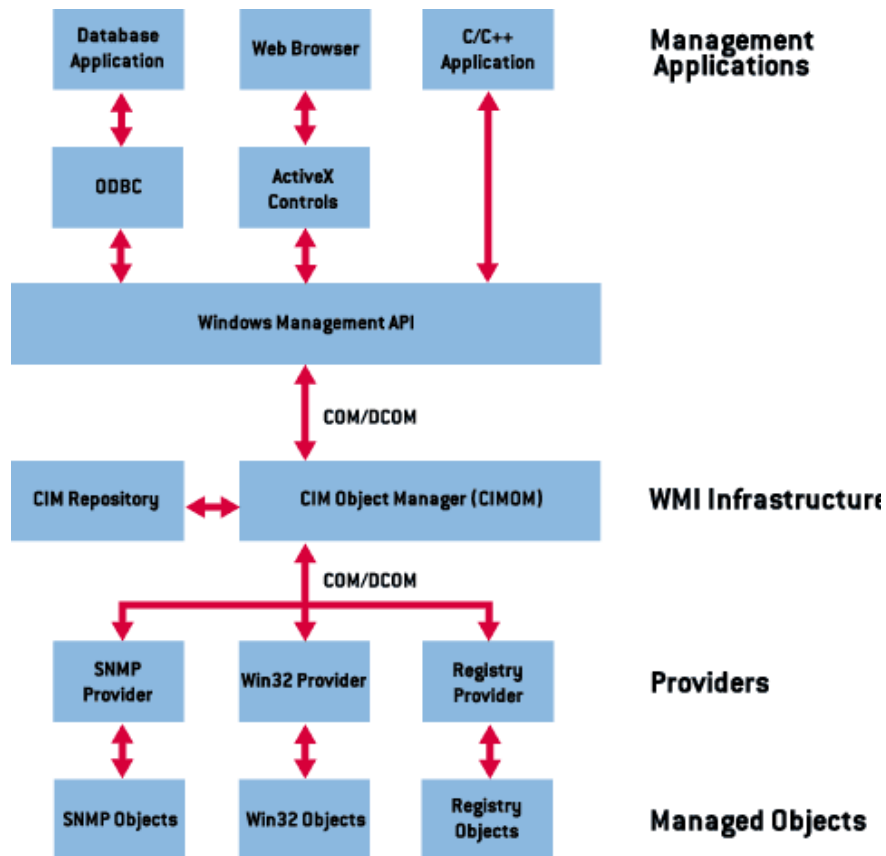
WMI Architecture

WMI architecture consists of the four segments that Figure 1, page 74, shows: management applications, WMI infrastructure, providers, and managed objects. Management applications are Windows applications that access and display or process data that the applications obtain about managed objects. A simple example of a management application is a Performance Monitor replacement that relies on WMI, rather than the Performance API, to obtain performance information. A more complex example is an enterprise-management tool that lets administrators perform automated inventories of the software and hardware configuration of every computer in their enterprise.

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



Developers typically must target management applications to collect data from and manage specific objects. An object might represent one component, such as a network adapter device, or a collection of components, such as a computer (the computer object might contain the network adapter object). Providers need to define and export the representation of the objects that management applications are interested in. For example, if the vendor of a network adapter wants the adapter to be WMI-capable, the vendor must develop a provider that serves as the interface to the adapter, querying and setting the attributes' state and behavior as the management applications direct. In some cases (e.g., device drivers), Microsoft supplies a provider that has its own API to help developers of provider plugins leverage the provider's implementation with minimal coding effort.

The WMI infrastructure is the glue that binds management applications and providers. The infrastructure also serves as the object-class store and, in many cases, as the storage manager for persistent object properties. WMI implements the store, or repository, as an on-disk database. As part of its infrastructure, WMI supports several APIs through which management applications access object data and providers supply data and class definitions.

Win32 programs use the WMI COM API, the primary management API, to directly interact with WMI. Other APIs layer on top of the COM API and include an ODBC adapter for Microsoft's Access database application. A database developer uses the WMI ODBC adapter to embed references to object data in the developer's database. Then, the developer can easily generate reports with database queries that contain WMI-based data. WMI ActiveX controls support another layered API. Web developers use the ActiveX controls to construct Web-based interfaces to WMI data. Another management API is the WMI scripting API, for use in script-based applications and Visual Basic (VB) programs. WMI scripting support exists for VBScript, JScript, Active Server Pages (ASP), and HTML.

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

As they are for management applications, WMI COM interfaces constitute the primary API for providers. However, unlike management applications, which are COM clients, providers are COM or Distributed COM (DCOM) servers (i.e., the providers implement COM objects that WMI interacts with). Possible embodiments of a WMI provider include DLLs that load into WMI's manager process, and standalone Win32 applications or Win32 services. Microsoft includes a number of built-in providers that present data from well-known sources such as the Performance API, the Registry, and the Event Manager. The WMI software development kit (SDK) lets developers develop third-party WMI providers.

Providers

At WBEM's core is the DMTF-designed Common Information Model (CIM) specification. CIM specifies how management systems represent, from a systems management perspective, anything from a computer to an application or device on a computer. Provider developers use CIM to represent the components that make up the parts of an application for which the developers want to enable management. Developers use the Managed Object Format (MOF) language to implement a CIM representation.

In addition to defining objects, a provider must interface WMI to the objects. WMI classifies providers according to the interface features the providers supply. Table 1 lists WMI provider classifications. Note that a provider can implement one or more features; therefore, a provider can be, for example, both a class and an event provider. To clarify the feature definitions in Table 1, let me discuss a provider that implements most of those features. The Event Log provider defines several objects, including an Event Log Computer, Event Log Record, and Event Log File. The Event Log provider is a Class provider because it defines these objects using classes and must give the class definitions to WMI. This provider is also an instance provider because it can define multiple instances for several of its classes. One class for which the Event Log provider defines multiple instances is the Event Log File class; the Event Log provider defines an instance for each of the system's event logs (i.e., System Event Log, Application Event Log, and Security Event Log).

TABLE 1: Provider Classifications

Classification	Description
Class	Can retrieve, modify, delete, and enumerate a provider-specific class. Can also support query processing.
Instance	Can retrieve, modify, delete, and enumerate instances of system and provider-specific classes. Can also support query processing.
Property	Can retrieve and modify individual property values.
Method	Invokes methods for a provider-specific class.
Event	Generates events notifications.
Event Consumer	Maps a physical consumer to a logical consumer to support event notification.

The Event Log provider defines the instance data and lets management applications enumerate the records. The Event Log provider also lets a management application query specific Event Log records' properties; this capability classifies the Event Log provider as a Property provider. To let management applications use WMI to back up and restore the Event Log files, the Event Log provider

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

implements backup and restore methods for Event Log File objects. Doing so makes the Event Log provider a Method provider. Finally, a management application can register to receive notification whenever a new record writes to one of the Event Logs. Thus, the Event Log provider serves as an Event provider when it uses WMI event notification to tell WMI that Event Log records have arrived.

CIM and MOF

CIM follows in the steps of object-oriented languages such as C++ and Java, in which a modeler designs representations as classes. Using classes lets developers use the powerful modeling techniques of inheritance and composition. Subclasses can inherit the attributes of another class and add their own characteristics or override the characteristics they inherit from the parent class. A class that inherits properties from another class derives from the parent class. Classes also compose: A developer can build a class that includes other classes.

The DMTF provides multiple classes as part of the WBEM standard. These classes are CIM's basic language and represent objects that apply to all areas of management. The classes are part of the CIM core model. An example of a core class is `CIM_ManagedSystemElement`. This class contains a few basic properties that identify physical components such as hardware devices and logical components such as processes and files. The properties include a caption, description, installation date, and status. Thus, the `CIM_LogicalElement` and `CIM_PhysicalElement` classes inherit the attributes of the `CIM_ManagedSystemElement` class. These two classes are also part of the CIM core model. The WBEM standard calls these classes abstract classes because they exist solely as classes that other classes inherit (i.e., no object instances of an abstract class exist). You can therefore think of abstract classes as templates that define properties for use in other classes.

A second category of classes represents objects that are specific to management areas but independent of a particular implementation. These classes constitute the common model and are considered an extension of the core model. An example of a common-model class is the `CIM_FileSystem` class, which inherits the attributes of `CIM_LogicalElement`. Because virtually every OS, including Win2K, Linux, and other varieties of UNIX, rely on file-system-based structured storage, the `CIM_FileSystem` class is an appropriate constituent of the common model.

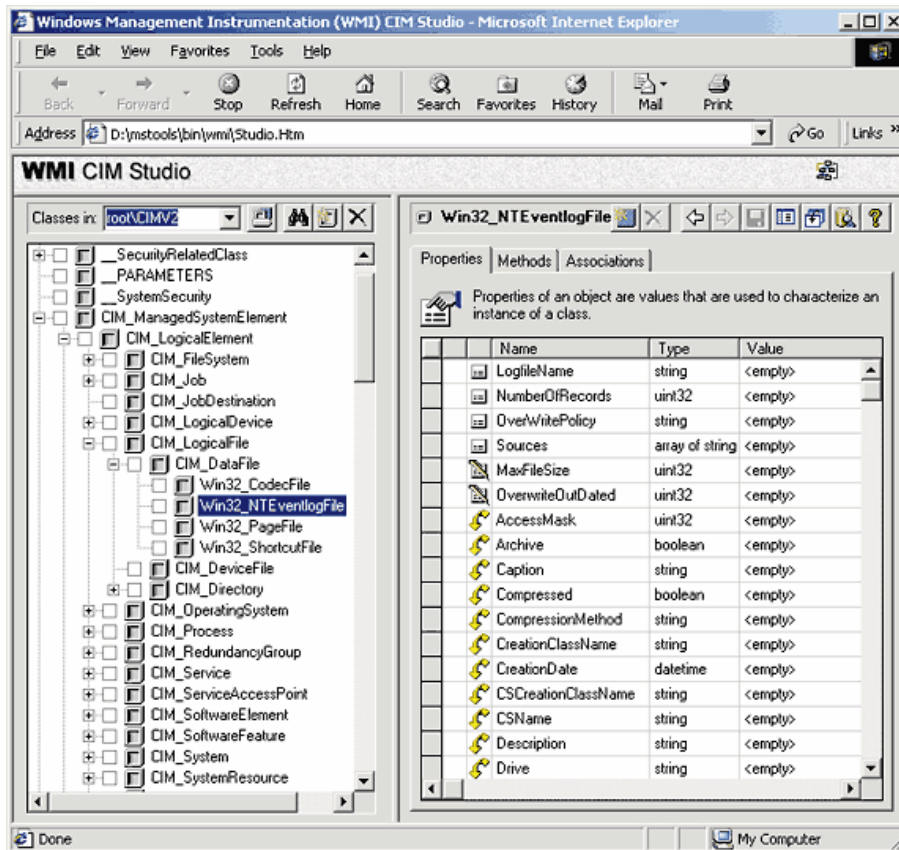
The final class category comprises technology-specific additions to the common class. Win2K defines a large set of these classes to represent objects specific to the Win32 environment. Because all OSs store data in files, the CIM common model includes the `CIM_LogicalFile` class. The `CIM_DataFile` class inherits the `CIM_LogicalFile` class, and Win32 adds the `Win32_PageFile` and `Win32_ShortCutFile` file classes for those Win32 file types.

The Event Log provider makes extensive use of inheritance. Screen 1 shows a view of the WMI CIM Studio, a class browser that ships with the WMI SDK. (Microsoft supplies the WMI SDK with Microsoft Developer Network—MSDN—software.) You can see where the Event Log provider relies on inheritance in the provider's `Win32_NTEventlogFile` class, which derives from `CIM_DataFile`. Event Log files are data files that have additional Event Log-specific attributes such as a log file name and a count of the number of records that the file contains. The tree that the class browser shows reveals that `Win32_NTEventlogFile` is based on several levels of inheritance, in which `CIM_DataFile` derives from `CIM_LogicalElement`, and `CIM_LogicalElement` derives from `CIM_ManagedSystemElement`.

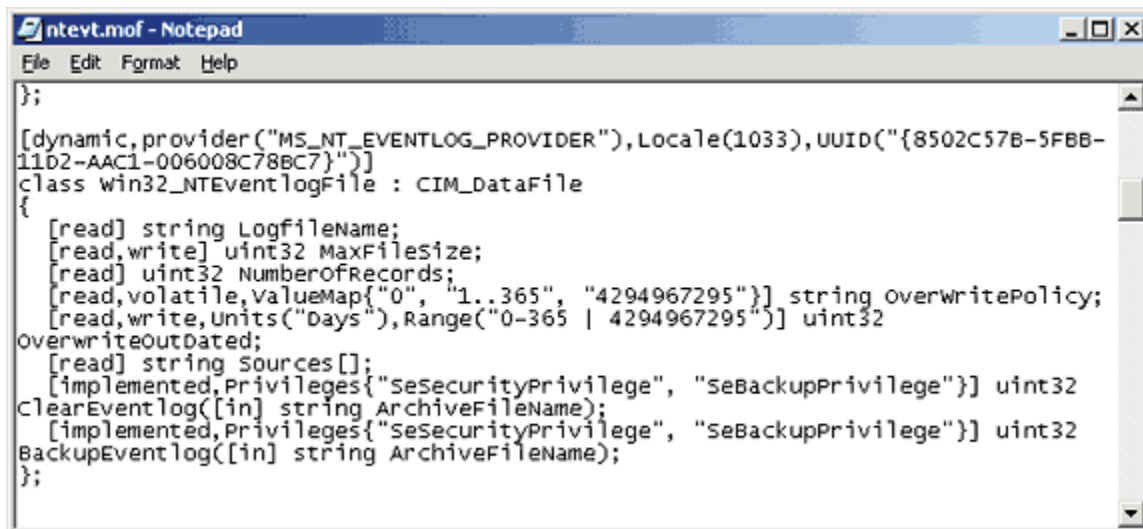
Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



As I stated earlier, WMI class-provider developers write their classes in MOF. Screen 2, page 78, shows the definition of the Event Log provider's Win32_NTEventlogFile, which is selected in Screen 1. Note the correlation between the first six properties that the right panel lists in Screen 1 and those properties' definitions in the MOF file in Screen 2. CIM Studio uses yellow arrows to tag those properties that a class inherits. Thus, you don't see those properties specified in Win32_NTEventlogFile's definition.



Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

One term worth reviewing is dynamic, which is a descriptive designator for the Win32_NTEventlogFile class that the MOF file in Screen 2 shows. The term means that the WMI infrastructure asks the WMI provider for the values of properties associated with a class whenever a management application queries the properties. A static class is one for which the provider stores the properties in the WMI repository; the WMI infrastructure refers to the repository to obtain the values instead of asking the provider for the values. Because updating the repository is a relatively expensive operation, dynamic providers are more efficient for objects that have properties that change frequently.

After constructing classes in MOF, WMI developers can supply the class definitions to WMI in several ways. One way is for WMI developers to compile an MOF file into a binary MOF (BMF) file—a more compact binary representation than an MOF file—and give the BMF files to the WMI infrastructure. Another way is for the provider to compile the MOF and use WMI COM APIs to give the definitions to the WMI infrastructure. Finally, a provider can use the mofcomp tool to directly give the WMI infrastructure a classes-compiled representation.

The WMI Namespace

Classes define the properties of objects, and class instances represent objects on a system. WMI uses a namespace that contains several subnamespaces that WMI arranges hierarchically to organize object instances. A management application must connect to a namespace before the application can access objects within the namespace.

WMI names the namespace root directory root. All WMI installations have four predefined namespaces that reside beneath root: CIMV2, Default directory, Security, and WMI. Some of these namespaces have other namespaces within them. For example, CIMV2 includes the Applications and ms_409 namespaces as subnamespaces. Providers sometimes define their own namespaces; you can see the WMI namespace (which the Windows device driver WMI provider defines) beneath root on Win2K.

Unlike a file-system namespace, which comprises a hierarchy of directories and files, a WMI namespace is only one level deep. Instead of using names as a file system does, WMI uses object properties that it defines as keys to identify the objects. Management applications specify class names with key names to locate specific objects within a namespace. Thus, each instance of a class must be uniquely identifiable by its key values. For example, the Event Log provider uses the Win32_NTLogEvent class to represent records in an event log. This class has two keys, LogFile and RecordNumber, both of which are strings. A management application that queries WMI for instances of Event Log records obtains from the provider key pairs that identify records. The application refers to a record using the syntax that you see in this example object pathname:

```
\\MARKLAPTOP\CIMV2:Win32_NTLogEvent.Logfile="Application",RecordNumber="1"
```

The first component in the name identifies the computer on which the object is located, and the second component is the namespace in which the object resides. The class name follows the colon, and key names and their associated values follow the period. A comma separates the key values.

Class Association

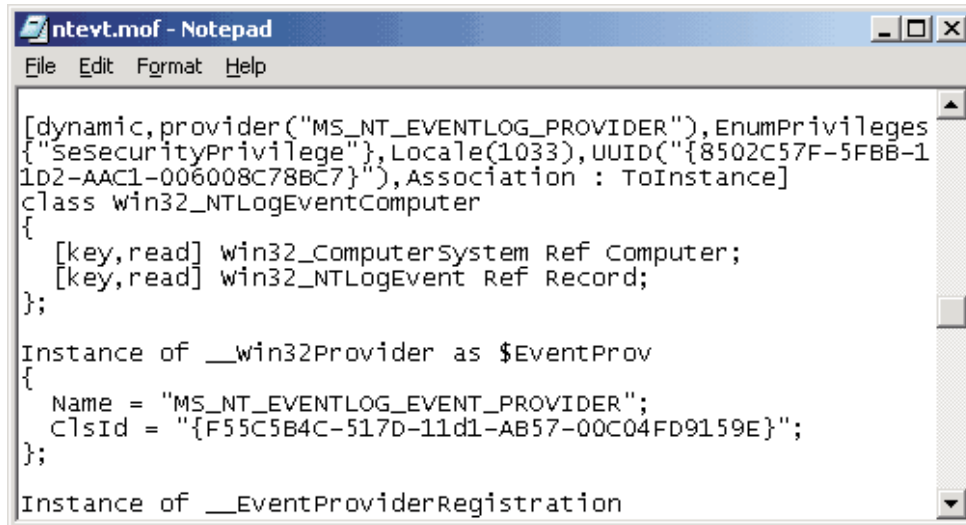
Many object types are related to one another in some way. For example, a computer object has a processor, installed software, an OS, active processes, and so on. WMI lets providers construct an association class to represent a logical connection between two different classes. Association classes associate one class with another, so the classes have only two properties. Because the properties are references to classes, the properties consist of a class name and the Ref modifier. Screen 3 shows an

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

association in which the Event Log provider's MOF file associates the Win32_NTLogEvent class with the Win32_ComputerSystem class. Given an object, a management application can query associated objects. In this way, a provider defines a hierarchy of objects.

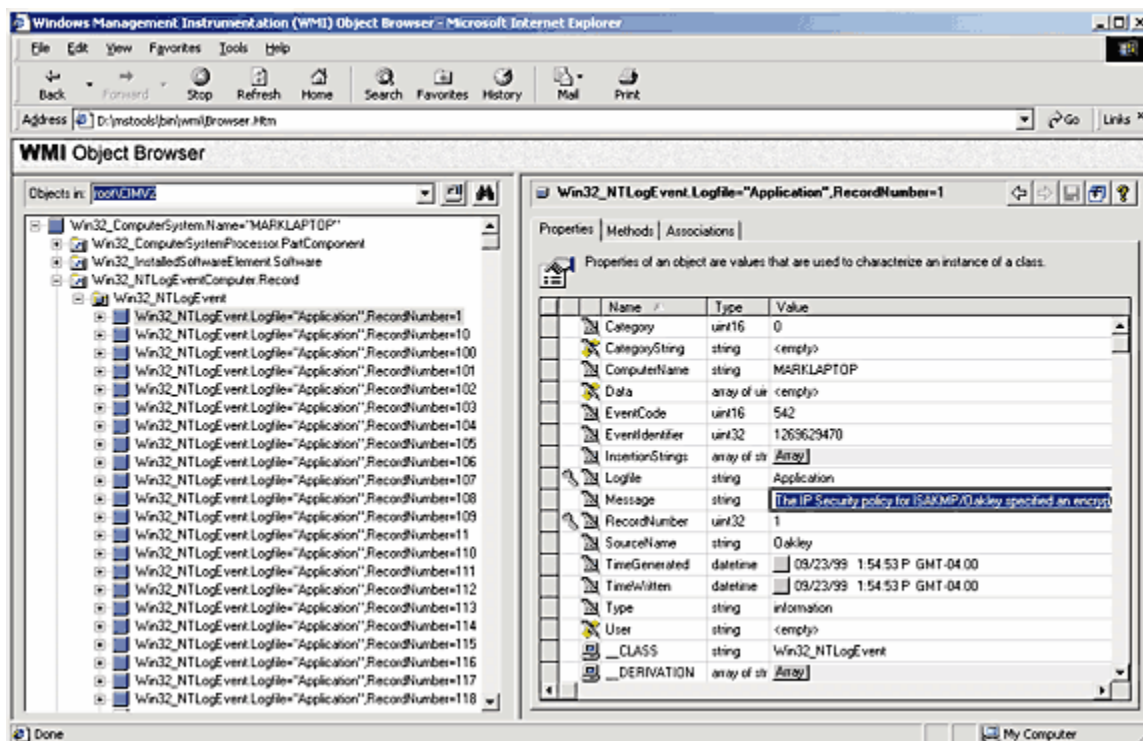


```
[dynamic,provider("MS_NT_EVENTLOG_PROVIDER"),EnumPrivileges
{"SeSecurityPrivilege"},Locale(1033),UUID("{8502C57F-5FBB-1
1D2-AAC1-006008C78BC7}"),Association : ToInstance]
class Win32_NTLogEventComputer
{
    [key,read] win32_ComputerSystem Ref Computer;
    [key,read] win32_NTLogEvent Ref Record;
};

Instance of __win32Provider as $EventProv
{
    Name = "MS_NT_EVENTLOG_EVENT_PROVIDER";
    CLSID = "{F55C5B4C-517D-11d1-AB57-00C04FD9159E}";
};

Instance of __EventProviderRegistration
```

Screen 4, page 80, shows the WMI Object Browser (another development tool that the WMI SDK includes) viewing the root of the CIMV2 namespace. Win32 system components typically place their objects within the CIMV2 namespace. Object Browser first locates the Win32_ComputerSystem object instance "MARKLAPTOP", which is the object that represents the computer. Then, Object Browser obtains the objects associated with Win32_ComputerSystem and displays them beneath "MARKLAPTOP". The Object Browser user interface (UI) displays association objects with a double-arrow folder icon. The associated class type's objects display beneath the folder.



Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

You can see in Object Browser that the Event Log provider's association class Win32_NTLogEventComputer is beneath "MARKLAPTOP", and that numerous instances of the Win32_NTLogEvent class exist. Refer to Screen 3 to verify that the MOF file defines the Win32_NTLogEventComputer class to associate the Win32_ComputerSystem class with the Win32_NTLogEvent class. Selecting an instance of Win32_NTLogEvent in Object Browser reveals that class's properties in the right-hand pane. Microsoft intended Object Browser to help WMI developers examine their objects, but a management application would perform the same operations and display properties or collected information in a more intelligible manner.

WMI Implementation

The WMI infrastructure implements primarily in the `\winnt\system32\wbem\winmgmt.exe` executable file. This file runs as a Win32 service that the Win2K Service Control Manager starts the first time a management application or WMI provider tries to access WMI APIs. Most WMI components reside in `\winnt\system32` and `\winnt\system32\wbem`, including Win32 MOF files, built-in provider DLLs, and management application WMI DLLs. Look in the `\winnt\system32\wbem` directory, and you'll find `ntevt.mof`, the Event Log provider MOF file. You'll also find `ntevt.dll`, the Event Log provider's DLL, which `winmgmt.exe` loads.

Directories beneath `\winnt\system32\wbem` store the repository, log files, and third-party MOF files. WMI implements the repository—named the CIM Object Management (CIMOM) repository—as the file `\winnt\system32\wbem\repository\cim.rep`. WinMgmt honors numerous Registry settings related to the repository (including various internal performance parameters such as CIMOM backup locations and intervals) that the repository's `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM` Registry key stores.

Microsoft wanted to extend management capabilities to all aspects of Win2K, so the company had to create a way for device drivers to interact with WMI. Device drivers use several new interfaces to provide data to and accept commands from WMI. Microsoft calls the WMI commands System Control commands. The company assigned the name Windows Driver Model (WDM) Provider to the device driver provider because the same WMI interfaces in Win2K drivers exist for Win98 drivers. Because the interfaces are cross-platform, they fall under WDM, the cross-platform device driver architecture. Win2K places WDM objects in the `\root\wmi` namespace.

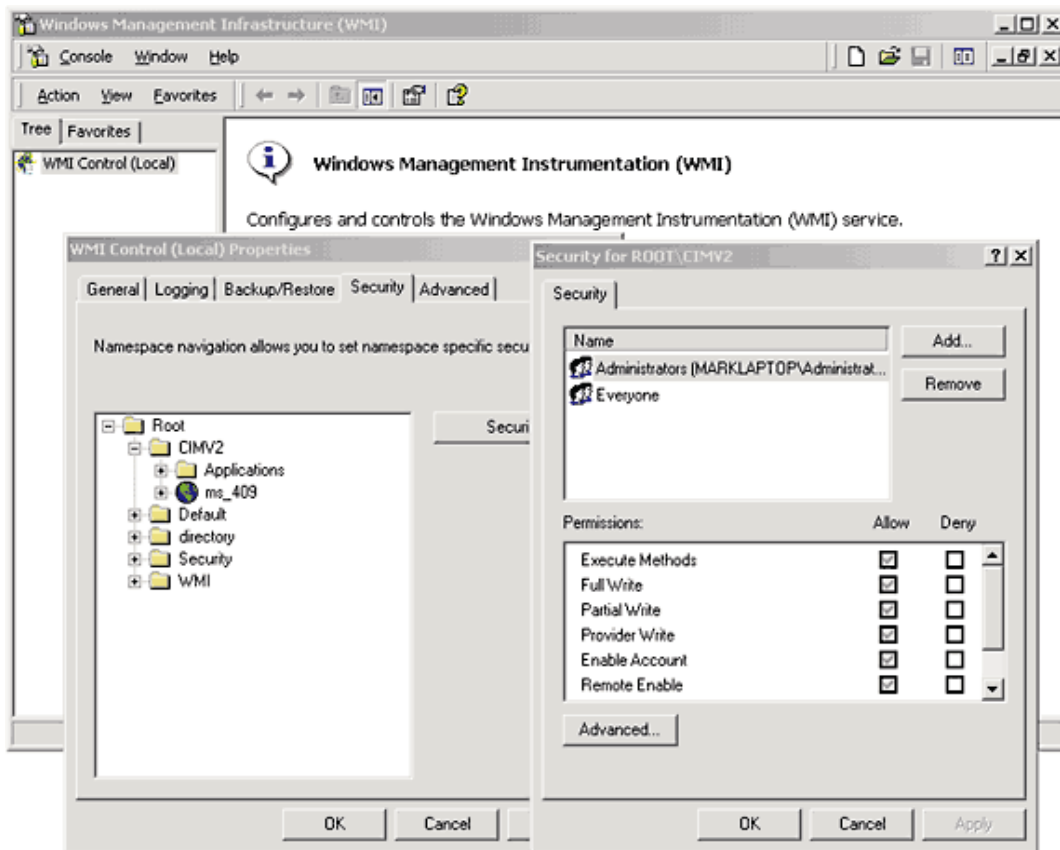
WMI Security

WMI implements security at the namespace level. If a management application successfully connects to a namespace, the application can view and access the properties of all the objects in that namespace. An administrator can use the WMI Control application to control which users can access a namespace. To start the WMI control application, from the Start menu, select Administrative Tools, Computer Management. Next, open the Services and Applications branch. Right-click WMI Control, and select Properties to launch the WMI Control (Local) Properties dialog box, which Screen 5 shows. To configure security for namespaces on the Security tab, select the namespace and click Security. Other tabs in the dialog box let you modify the performance and backup settings that the Registry stores.

Inside Windows Management Interface

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



Enterprise Management Comes to NT

Although I've covered a lot of ground in a short space, I've only skimmed the surface of WMI's rich functionality. WMI's flexibility and universal application throughout Win2K make it a powerful middleware technology that you'll see enterprise tool vendors take advantage of to deliver powerful enterprise-management tools.