Solaris WBEM SDK Developer's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 806–6828–10
May 2002

020123@3062

Contents

Preface 13

1 Overview of Solaris Web-Based Enterprise Management 17
About Web-Based Enterprise Management 17
About the Common Information Model 18
About Solaris WBEM Services 18
CIM Object Manager 19
Managed Object Format Compiler 19
Solaris Schema 19
Solaris WBEM SDK 20
Using CIM Workshop to Develop WBEM Applications 21
CIM Workshop Documentation 22
Running CIM Workshop 22

2 Creating JavaBeans Using the MOF Compiler 25
About The MOF Compiler 25
Generating JavaBeans Using `mofcomp` 26
How CIM Maps to Java 27
Generating JavaBeans Example 31

3 Writing a Client Program 45
Overview 45
Sequence of a Client Application 45
Opening and Closing a Client Connection 46
About Namespaces 46

Opening a Client Connection 47
Closing a Client Connection 48
Performing Basic Client Operations 48
Creating an Instance 49
Deleting an Instance 49
Getting and Setting Instances 51
Getting and Setting Properties 52
Enumerating Objects 53
Creating Associations 59
Calling Methods 62
Retrieving Class Definitions 63
Handling Exceptions 64
Creating a Namespace 65
Deleting a Namespace 66
Creating a Base Class 66
Deleting a Class 66
Setting Access Control 68
The `Solaris_UserAcl` Class 68
The `Solaris_NamespaceAcl` Class 70
Working with Qualifiers and Qualifier Types 71
Getting and Setting CIM Qualifiers 71
Batching Client Requests 72
Handling CIM Events 74
About Indications 75
About Subscriptions 77
Adding a CIM Listener 77
Creating an Event Filter 78
Creating an Event Handler 80
Binding an Event Filter to an Event Handler 81
Reading and Writing Log Messages 82
About Log Files 82

4 Writing a Provider Program 89
About Providers 89
Provider Data Sources 90
Types of Providers 91
Implementing the Provider Interfaces 93

Writing an Instance Provider 93
Writing a Method Provider 96
Writing an Associator Provider 97
Writing an Indication Provider 99
Writing a Native Provider 102
Creating a Provider 103
▼ How to Set the Provider `CLASSPATH` 103
▼ How to Register a Provider 104

5 Writing WBEM Queries 107
About the WBEM Query Language 107
Writing Queries 108
WQL Key Words 108
Parsing Queries 111
SELECT List 111
FROM Clause 112
WHERE Clause 112
Writing a Provider That Handles Queries 113

6 Using the Solaris WBEM SDK Sample Programs 115
About the Sample Programs 115
Sample Applet 116
▼ How to Run the Sample Applet Using Appletviewer 116
▼ How to Run the Sample Applet in a Web Browser 116
Sample Client Programs 116
Running the Sample Client Programs 118
Sample Provider Programs 118
▼ How to Run the Sample Provider Programs 119

A WBEM Error Messages 121
About WBEM Error Messages 121
Parts of an Error Message 121
List of Error Messages 122

B The Solaris Schema 137
Solaris Schema Files 138

`Solaris_Schema1.0.mof` File 139
`Solaris_CIMOM1.0.mof` File 140
`Solaris_Core1.0.mof` File 140
`Solaris_Application1.0.mof` File 140
`Solaris_System1.0.mof` File 141
`Solaris_Device1.0.mof` File 142
`Solaris_Acl1.0.mof` File 143
`Solaris_Network1.0.mof` File 143
`Solaris_Users1.0.mof` File 143
`Solaris_Event1.0.mof` File 143
`Solaris_SNMP1.0.mof` File 144
`Solaris_VM1.0.mof` File 144
`Solaris_Project1.0.mof` File 145
`Solaris_Performance1.0.mof` File 146

Index 147

Tables

TABLE 1–1 Solaris WBEM APIs 20
TABLE 2–1 MOF File Elements 27
TABLE 2–2 How CIM Elements Map to Java Elements 27
TABLE 2–3 How CIM Data Types Map to Java Data Elements 28
TABLE 2–4 Meta Qualifiers 29
TABLE 2–5 Standard Qualifiers 29
TABLE 2–6 How MOF Elements Map to Java Elements 31
TABLE 3–1 Enumerating Objects 54
TABLE 3–2 Association Methods 59
TABLE 3–3 TeacherStudent Methods 60
TABLE 3–4 `invokeMethod` Parameters 62
TABLE 3–5 `CIM_Indication` Class Structure 75
TABLE 3–6 `CIM_IndicationFilter` Properties 78
TABLE 3–7 `CIM_IndicationHandler` Properties 80
TABLE 3–8 Log Message Elements 82
TABLE 4–1 Provider Types 91
TABLE 4–2 `EventProvider` Methods 100
TABLE 5–1 Mapping of SQL Concepts to WQL 107
TABLE 5–2 Supported WQL Key Words 108
TABLE 5–3 Sample SELECT Statements 109
TABLE 5–4 WQL Operators for WHERE Clauses 111
TABLE 6–1 Sample Client Programs 117
TABLE 6–2 Sample Provider Programs 119
TABLE B–1 Solaris Schema Files 138

Figures

FIGURE 3–1 TeacherStudent Association 1 59
FIGURE 3–2 TeacherStudent Association 2 60

Examples

EXAMPLE 2–1 Generating JavaBeans 31
EXAMPLE 3–1 Connecting to the Root Account 47
EXAMPLE 3–2 Connecting to a User Account 47
EXAMPLE 3–3 Authenticating as an RBAC Role Identity 48
EXAMPLE 3–4 Closing a Client Connection 48
EXAMPLE 3–5 Creating an Instance 49
EXAMPLE 3–6 Deleting Instances 50
EXAMPLE 3–7 Getting and Setting Instances 51
EXAMPLE 3–8 Getting a Property 52
EXAMPLE 3–9 Setting a Property 53
EXAMPLE 3–10 Enumerating Classes 54
EXAMPLE 3–11 Enumerating Classes and Instances 55
EXAMPLE 3–12 Enumerating Class Names 57
EXAMPLE 3–13 Enumerating Namespaces 57
EXAMPLE 3–14 Passing Instances 62
EXAMPLE 3–15 Calling a Method 63
EXAMPLE 3–16 Retrieving a Class Definition 63
EXAMPLE 3–17 Creating a Namespace 65
EXAMPLE 3–18 Deleting a Class 67
EXAMPLE 3–19 Setting CIM Qualifiers 71
EXAMPLE 3–20 Batching Example 73
EXAMPLE 3–21 Adding a CIM Listener 78
EXAMPLE 3–22 Creating an Event Filter 80
EXAMPLE 3–23 Creating an Event Handler 81
EXAMPLE 3–24 Binding an Event Filter to an Event Handler 81
EXAMPLE 3–25 Creating an Instance of `Solaris_LogEntry` 83

EXAMPLE 3–26 Displaying a List of Log Records 85
EXAMPLE 4–1 `CIMInstance` Provider 94
EXAMPLE 4–2 Method Provider 96
EXAMPLE 4–3 `CIMAssociator` Provider 98
EXAMPLE 4–4 Registering a Provider 105
EXAMPLE 5–1 Provider That Handles Queries 113
EXAMPLE A–1 Parts of an Error Message 121

Preface

The *Solaris WBEM SDK Developer's Guide* describes the Solaris™ Web-Based Enterprise Management Software Developer's Kit (SDK), which enables developers to create standards-based applications that manage resources in the Solaris™ operating environment. Developers can also use this toolkit to write providers, which are programs that communicate with managed resources to access data.

The Solaris WBEM SDK includes client application programming interfaces (APIs) for describing and managing resources in the Distributed Management Task Force (DMTF) Common Information Model (CIM), and provider APIs for getting and setting dynamic data on managed resources. The Solaris WBEM SDK also includes CIM Workshop, a Java application that you can use to create and view managed resources on a system, and sample WBEM client and provider programs.

Who Should Use This Book

This book is for the following types of software developers:

- **Instrumentation developers** – Communicate device information in a standard CIM format to the CIM Object Manager through software providers.
- **System and network application developers** – Write applications that manage the information stored in CIM classes and instances, and use the Solaris WBEM Services APIs to get and set the properties of CIM instances and classes.

Before You Read This Book

This book requires a solid understanding of the following:

- Object-oriented programming concepts
- Java programming
- Common Information Model (CIM) concepts

If you are unfamiliar with these areas, you might find the following references useful:

- *Java: How to Program*, H. M. Deitel and P. J. Deitel, Prentice Hall, ISBN 0–13–263401–5.
- *The Java Class Libraries, Second Edition, Volume 1*, Patrick Chan, Rosanna Lee, Douglas Kramer, Addison-Wesley, ISBN 0–201–31002–3.
- *CIM Tutorial*, provided by the Distributed Management Task Force (DMTF).

The following web sites are useful resources for WBEM technologies:

- Distributed Management Task Force (DMTF) –This site includes the latest CIM developments, information about industry working groups, and information on how to extend the CIM Schema.
- Rational Software – This site contains documentation on the Unified Modeling Language (UML).

How This Book Is Organized

Chapter 1 introduces Web-Based Enterprise Management (WBEM), the Common Information Model (CIM), the Solaris WBEM SDK Application Programming Interfaces (APIs), and CIM Workshop.

Chapter 2 explains how to use the MOF compiler.

Chapter 3 explains how to use the client APIs to write client programs.

Chapter 4 explains how to use the provider APIs to write provider programs.

Chapter 5 explains how to use the Query APIs and the WBEM Query Language (WQL) to write and handle queries.

Chapter 6 describes the sample programs provided with the Solaris WBEM SDK.

Appendix A explains the error messages generated by components of the Solaris WBEM SDK.

Appendix B describes the MOF files included with the Solaris WBEM SDK.

Related Information

You might also want to refer to the following related documentation:

- *Solaris WBEM Services Administration Guide* – Describe how to administer Web-based Enterprise Management (WBEM) services in the Solaris operating environment.
- Javadoc™ reference pages – Describe the WBEM APIs. See `file:/usr/sadm/lib/wbem/doc/index.html`.
- CIM/Solaris Schema – Describe the CIM and Solaris Schema. See `file:/usr/sadm/lib/wbem/doc/mofhtml/index.html`.
- Distributed Management Task Force (DMTF) Glossary – A comprehensive glossary of CIM and WBEM-related terms. See `http://www.dmtf.org/education/cimtutorial/reference/glossary.php`.

Accessing Sun Documentation Online

The docs.sun.com℠ Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Overview of Solaris Web-Based Enterprise Management

This chapter provides an overview of Solaris Web-Based Enterprise Management, and includes the following topics:

- "About Web-Based Enterprise Management" on page 17
- "About the Common Information Model" on page 18
- "About Solaris WBEM Services" on page 18
- "Using CIM Workshop to Develop WBEM Applications" on page 21

Note – This chapter provides a general overview of Web-Based Enterprise Management (WBEM) and the Common Information Model (CIM). For more in-depth information about WBEM and CIM, refer to the Distributed Management Task Force's (DMTF) website at `http://www.dmtf.org`.

About Web-Based Enterprise Management

Web-Based Enterprise Management (WBEM) is a set of management and Internet technologies that unify the management of enterprise computing environments. With WBEM, you can deliver an integrated set of standardized management tools that leverage emerging web technologies. By developing management applications according to WBEM principles, you can create compatible products at a low development cost.

The Distributed Management Task Force (DMTF), an industry group that represents corporations in the computer and telecommunications industries, is leading the effort to develop and disseminate standards for the management of desktop environments,

enterprise-wide systems, and the Internet. The goal of the DMTF is to develop an integrated approach to managing computers and networks across platforms and protocols, resulting in cost-effective products that interoperate as flawlessly as possible.

About the Common Information Model

The Common Information Model (CIM), developed by the DMTF, is an industry standard used to manage systems and networks. This standard provides a common conceptual framework that classifies and defines the parts of a networked environment and depicts how these various parts interact. The CIM captures notions that are applicable to all areas of management, independent of technology implementation.

CIM consists of the following:

- **CIM Specification** – Defines the language and methodology for integration with other management models.
- **CIM Schema** – Provides the actual model descriptions for systems, applications, local area networks, and devices. The CIM Schema consists of the following:
  - **Core Model** – Provides the underlying, general assumptions of the managed environment. This model comprises a small set of classes and associations that provide a basic vocabulary for analyzing and describing managed systems.
  - **Common Model** – Captures notions that are common to particular management areas, but independent of a particular technology or implementation. Provides a basis for the development of management applications.
- **Extension schema** – Represents technology and platform-specific extensions to the Common Model. These schemas are specific to environments, such as operating systems. For example, the Solaris Schema is an extension schema. Vendors extend the model for their products by creating subclasses of objects. Applications can then transverse object instances in the standard model to manage different products in a heterogeneous environment.

About Solaris WBEM Services

Solaris WBEM Services is the Solaris implementation of WBEM and CIM standards. The following components are included with Solaris WBEM Services:

- "CIM Object Manager" on page 19

- "Managed Object Format Compiler" on page 19
- "Solaris Schema" on page 19
- "Solaris WBEM SDK" on page 20

CIM Object Manager

The CIM Object Manager manages CIM objects on a WBEM-enabled system. When a WBEM client application accesses information about a CIM object, the CIM Object Manager contacts either the appropriate provider for that object, or the CIM Object Manager Repository. When a WBEM client application requests data from a managed resource that is not available for the CIM Object Manager Repository, the CIM Object Manager forwards the request to the provider for that managed resource. The provider dynamically retrieves the information.

WBEM client applications contact the CIM Object Manager to establish a connection to perform WBEM operations, such as creating a CIM class or updating a CIM instance. When a WBEM client application connects to the CIM Object Manager, the WBEM client gets a reference to the CIM Object Manager, which it then uses to request services and perform operations.

Managed Object Format Compiler

You use the Managed Object Format (MOF) language to specify CIM schema. You define classes and instances using ASCII text, and place them in a file that you submit to the MOF compiler, `mofcomp`(1M). The MOF compiler parses the file and adds the classes and instances defined in the file to the CIM Object Manager repository. See Chapter 2 for information on how to use the MOF compiler to automatically generate JavaBeans from MOF files.

Because you can convert MOF to Java, applications developed in MOF can run on any system or in any environment that supports Java.

Note – For more in-depth information about the MOF language, files, and syntax, see http://www.dmtf.org/education/cimtutorial/extend/spec.php.

Solaris Schema

An extension schema of the Common Model, the Solaris Schema specifically describes managed objects running in the Solaris operating environment.

When you install Solaris WBEM Services, the CIM Schema and the Solaris Schema MOF files populate the `/usr/sadm/mof` directory. These files are automatically compiled when the CIM Object Manager starts. The CIM Schema files, denoted by the `CIM_` prefix, form standard CIM objects. The Solaris Schema extends the standard CIM Schema by describing Solaris objects. The MOF files that make up the Solaris Schema have the `Solaris_` prefix.

Note – The CIM and Solaris Schema is installed at `file:/usr/sadm/lib/wbem/doc/mofhtml/index.html`.

Solaris WBEM SDK

The Solaris WBEM SDK is a set of APIs that contain the components necessary to write management applications that communicate with WBEM-enabled management devices using XML and HTTP communication standards.

Solaris WBEM applications request information or services from the Common Information Model (CIM) Object Manager through the WBEM APIs. These APIs represent CIM objects as Java classes. You use the APIs to describe managed objects and retrieve information about managed objects in a system environment. The advantage of modeling managed resources using CIM is that those objects can be shared across any system that is CIM-compliant.

Note – The Solaris WBEM Application Programming Interface (API) documentation is in Javadoc format and is installed at `file:/usr/sadm/lib/wbem/doc/index.html` during a Solaris installation.

The Solaris WBEM APIs are described in the following table:

TABLE 1–1 Solaris WBEM APIs

API	Package Name	Description
CIM	`javax.wbem.cim`	Includes common classes and methods that represent the basic CIM elements. The CIM APIs create objects on the local system.

TABLE 1–1 Solaris WBEM APIs *(Continued)*

API	Package Name	Description
Client See Chapter 3.	`java.wbem.client`	Applications use the `CIMClient` class to connect to the CIM Object Manager, and use the other classes and methods to transfer data to and from the CIM Object Manager. The new Batching APIs, a subset of the Client APIs, enable clients to batch multiple requests in one remote call, reducing the delay introduced by multiple remote message exchanges.
Provider See Chapter 4.	`java.wbem.provider`	The CIM Object Manager uses these APIs to pass application requests for dynamic data to providers.
Query See Chapter 5.	`java.wbem.query`	Contains classes and methods that you use to formulate and manipulate queries using the WBEM Query Language (WQL).

Using CIM Workshop to Develop WBEM Applications

You can develop WBEM applications using CIM Workshop, a GUI-based development tool included with the Solaris WBEM SDK. You use CIM WorkShop to do the following:

- View, add, delete, and search for classes
- View, add, and delete namespaces
- Add properties, qualifiers, and methods to new classes
- Create instances
- Modify instance values
- Traverse associations
- Subscribe to events
- Execute methods

Note – CIM guidelines prevent you from modifying the properties, methods, and qualifiers of CIM Schema and Solaris Schema classes, and from changing the values of inherited properties, methods, and qualifiers.

CIM Workshop Documentation

CIM Workshop has context-sensitive online help for every dialog box except for the main window. When you click the various interface components such as text entry fields, tabs, and radio buttons within a dialog box, the appropriate help displays in the Information pane on the left side of the dialog box.

Tip – To close and reopen the Information pane, click the question mark button on the upper left corner of the dialog box.

Running CIM Workshop

By default, CIM Workshop connects to a CIM Object Manager running on the local host in the `root/cimv2` default namespace using the Remote Method Invocation (RMI) protocol. You can also point to a remote host that is running the CIM Object Manager.

▼ To Start CIM WorkShop

1. **At the system prompt, type:**

   ```
   % /usr/sadm/bin/cimworkshop
   ```

   The CIM Workshop Login dialog box displays.

2. **Follow the instructions in the context-sensitive help to fill in the fields in the Login dialog box. Then click OK.**

   The CIM Workshop main window displays.

▼ To Exit CIM WorkShop

- **From the CIM Workshop main window, select Workshop->Exit.**

  CIM Workshop exits.

About the Main Window

The CIM Workshop main window is divided into three panes:

- **Left pane** – Displays the class inheritance tree of the current namespace.
- **Right pane** – Displays the Properties, Methods, and Events tabs. When you select a class in the left pane, click one of these tabs in the right pane to get more information on the properties, methods, or events of the selected class.

- **Bottom pane** – Displays notification when events occur to which you are subscribed.

CHAPTER 2

Creating JavaBeans Using the MOF Compiler

This chapter provides an overview of the Managed Object Format (MOF) Compiler, and describes how to create JavaBeans using the `-j` option with the `mofcomp` command. This chapter covers the following topics.

- "About The MOF Compiler" on page 25
- "How CIM Maps to Java " on page 27
- "Generating JavaBeans Example" on page 31

Note – For more information on the MOF Compiler, see `mofcomp`(1M).

About The MOF Compiler

Managed Object Format (MOF) is a compiled language developed by the Distributed Management Task Force (DMTF) to define static and dynamic classes and instances for CIM and WBEM. You can use the CIM and Solaris MOFs that are included with Solaris WBEM Services, and you can create your own MOFs. For more information on creating your own MOFs using the DMTF's MOF language, see the DMTF Website at http://www.dmtf.org.

The *MOF Compiler*, `mofcomp`(1M), parses MOF files, converts the classes and instances to Java classes, and then adds the classes to the CIM Object Manager Repository in the default (`root\cimv2`) or other specified namespace. Because you can easily convert MOF files to Java, Java-based applications can interpret and exchange the data contained within MOF files on any machine that runs a Java Virtual Machine.

During a Solaris installation, the MOF Compiler automatically compiles the bundled MOF files that describe the CIM and Solaris Schema and adds them to the CIM Object Manager Repository.

Generating JavaBeans Using `mofcomp`

In the context of WBEM, JavaBeans™ or *beans*, define methods and instances for accessing and manipulating CIM classes and data elements. To simplify your development efforts, you can use the `-j` option with the `mofcomp` command to automatically generate JavaBeans that represent the CIM classes defined in your MOF files. These automatically-generated JavaBeans define the interfaces; it is up to you to add the implementation code.

Note – To safeguard your program from changes that you make to the underlying JavaBeans implementation, use the interfaces rather than the original JavaBeans.

When you specify the `-j` option with `mofcomp`, a base Java interface named `CIMBean.java` and a base bean that implements the interface named `CIMBeanImpl.java` is generated. `CIMBeanImpl.java` contains all of the code that is common to the generated beans. All generated Java interfaces inherit from `CIMBean.java` and all generated beans inherit from `CIMBeanImpl.java`, consequently inheriting the base implementation.

For each CIM class that is defined in a MOF file, The MOF Compiler JavaBeans Generation Feature generates a Java interface that contains the following:

- Accessor and mutator methods for the properties that are defined in the MOF file
- Methods that are comparable to the `invokeMethods` that are defined in the MOF file

The Java interfaces are named *CIMClass*`Bean.java`. Bean classes that implement those Java interfaces are named *CIMClass*`BeanImpl.java`. In addition, accessor methods for properties that contain the CIM `DisplayName`, `Units`, and `Version` qualifiers are generated.

For each `invokeMethod` that contains an `OUT` qualified parameter in a CIM class, a container interface that holds the output that the invoking of the method generates is generated. These interfaces are named *CIMClass_MethodName*`Output.java`. An instance of this *CIMClass_MethodName*`Output.java` container interface is required as the last parameter of the bean's method. This container interface is required because the Object datatype or datatypes that the bean's method takes as parameters are not mutatable, and therefore they cannot be used to hold both input and output data.

MOF File Elements

You must include the `PACKAGE` element in your MOF file to take advantage of the `-j` option. In addition, you can specify the `IMPORTS` and `EXCEPTIONS` elements, in the following format:

PACKAGE=*NameOfJavaPackage*
IMPORTS=*JavaClassName1*:*JavaClassName2*:...
EXCEPTIONS=*Exception1*:*Exception2*:...

Th following table describes these elements.

TABLE 2–1 MOF File Elements

Element	Description
PACKAGE	Required. Specifies the name of the Java package that contains the source files generated by the MOF Compiler.
IMPORTS	Optional. Specifies the name(s) of the Java classes to import into the generated source files, separated with a colon (:). You can specify as many Java classes as you want, on as many lines as you want.
EXCEPTIONS	Optional. Specifies the name(s) of the Java exceptions included in the generated source files, separated with a colon (:). You can specify as many Java class exceptions as you want, on as many lines as you want. **Note –** If you specify EXCEPTIONS, you must specify IMPORTS.

How CIM Maps to Java

The following table describes how CIM elements map to Java elements.

TABLE 2–2 How CIM Elements Map to Java Elements

CIM Element	Java Element
Class	The CIM class name is used as the basis for the name of the generated Java source files. The generated Java classes follow the same inheritance as defined in the class-subclass relationships in the MOF.
Property	An accessor and a mutator method is created for each CIM property. The CIM property name is used as the basis for the related accessor and mutator methods.

TABLE 2–2 How CIM Elements Map to Java Elements *(Continued)*

CIM Element	Java Element
Method	For each CIM method, a comparable Java method is created. The method name is used as the basis for the related Java method name. The return value is the same, accounting for the Java data type mapping. Input and output parameters are used as arguments to the Java method. Output parameters are not directly included in the method signature, but rather are encapsulated in an output container object which is included as a method parameter.
Qualifier	Qualifiers are described in Table 2–4 and Table 2–5.
Association	Nothing specific required.
Indication	Nothing specific required.
Reference	For each CIM reference, a reference to a generated Java interface is created.
Trigger	Nothing specific required.
Schema	Nothing specific required.

The following table describes how CIM data types map to Java data types.

TABLE 2–3 How CIM Data Types Map to Java Data Elements

CIM Data Type	Java Data Type	Accessor Method	Mutator Method
uint8 X	UnsignedInt8	UnsignedInt8 getX();	void setX(UnsignedInt8 x);
sint8 X	Byte	Byte getX();	void setX(Byte x);
uint16 X	UnsignedInt16	UnsignedInt16 getX();	void setX(UnsignedInt16 x);
sint16 X	Short	Short getX();	void setX(Short x);
uint32 X	UnsignedInt32	UnsignedInt32 getX();	void setX(UnsignedInt32 x);
sint32 X	Integer	Integer getX();	void setX(Integer x);
uint64 X	UnsignedInt64	UnsignedInt64 getX();	void setX(UnsignedInt64 x);
sint64 X	Long	Long getX();	void setX(Long x);
String X	String	String getX();	void setX(String x);
Boolean X	Boolean	Boolean isX();	void setX(Boolean x);
real32 X	Float	Float getX();	void setX(Float x);

TABLE 2–3 How CIM Data Types Map to Java Data Elements *(Continued)*

CIM Data Type	Java Data Type	Accessor Method	Mutator Method
real64 X	Double	Double getX();	void setX(Double x);
DateTime X	CIMDateTime	CIMDateTime getX();	void setX(CIMDateTime x);
Reference X	CIMObjectPath	CIMObjectPath getX();	void setX(CIMObjectPath x);
char16 X	Character	Character getX();	void setX(Character x);

The following table lists the meta qualifiers that refine the definition of the meta constructs in the model. These qualifiers are mutually exclusive and are used to refine the actual usage of an object class or property declaration within the MOF syntax.

TABLE 2–4 Meta Qualifiers

Qualifier	Scope	Type	Meaning
Association	class	Boolean	No affect on mapping
Indication	class	Boolean	Class is abstract

The following table lists the standard qualifiers and the affect that they have on the mapping of a CIM object to a bean. There is no support for optional qualifiers. JavaDoc is produced for each interface and class based on this mapping.

TABLE 2–5 Standard Qualifiers

Qualifier	Scope	Meaning
ABSTRACT	Class, Association, Indication	The class is abstract and has no effect on the Java interfaces
DESCRIPTION	Any	The information provided generates JavaDoc comments in the source file
DISPLAYNAME	Property	An accessor method for the display name is created: `public String displayNameFor`*`Property`*`();`
IN	Parameter	Determines the method signature
OUT	Parameter	Determines the Method Parameter signature and return values
TERMINAL	Class	Class or Interface is Final

TABLE 2–5 Standard Qualifiers *(Continued)*

Qualifier	Scope	Meaning
UNITS	Property, Method, Parameter	Another accessor method is created: `public String get`*property*`Units();`
VALUMAP	Property, Method, Parameter	Beans contain generated constants for each property in a CIM class that has a CIM ValueMap or a Values qualifier. How the constant name and constant value is obtained to generate these class variables depends on the data type of the property and which of the qualifiers the property possesses. **Note –** The ValueMap and Values qualifiers as defined in the CIM specification have meanings contrary to what the qualifier names might imply. ValueMap defines the legal set of values for a property and Values provides translation between an integer value and a string.
VALUES	Property, Method, Parameter	Beans contain generated constants for each property in a CIM class that has a CIM ValueMap or a Values qualifier. How the constant name and constant value is obtained to generate these class variables depends on the data type of the property and which of these qualifiers the property possesses. **Note –** The ValueMap and Values qualifiers as defined in the CIM specification have meanings contrary to what the qualifier names might imply. ValueMap defines the legal set of values for a property and Values provides translation between an integer value and a string.
VERSION	Class, Schema, Association, Indication	Class possesses a `getClassVersion()` method

The following table describes how MOF elements map to Java elements.

TABLE 2–6 How MOF Elements Map to Java Elements

MOF Element	Java Element
Description Qualifier	Description of the class, property, or method
Complete MOF representation of the class	The class JavaDoc description for both the Java interface and the implementing bean

Generating JavaBeans Example

Following is an example that shows the JavaBeans that are produced when you use the `mofcomp` command with the `-j` option.

You must run the `mofcomp` command as root or as a user with write access to the namespace in which you are compiling.

Note – Avoid specifying both the `-u` (user) `-p` (password) options when running the `mofcomp` command, as you must type in your password directly on the command line. Instead, specify only the `-u` option so that you are prompted to specify an encrypted password.

EXAMPLE 2–1 Generating JavaBeans

```
/usr/sadm/bin/mofcomp -u root -p mypassword -o /tmp
-j /tmp/bean.txt /usr/sadm/mof/Simple.mof
```

Following is the content of `/usr/sadm/mof/Simple.mof`:

```
/usr/sadm/mof/Simple.mof
-------------------
#pragma include ("CIM_Core26.mof")

class Simple_Class {

      [Key, Description ("Name of the class.") ]
   string Name;

      [Description ("Method to print the contents of the class.") ]
   string printClass();

};
```

Following is the content of `/tmp/bean.txt`:

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
/tmp/bean/txt
----------
PACKAGE=foo.com
IMPORTS=java.lang.Exception
EXCEPTIONS=Exception
```

Following is the content of `CIMBean.java`:

```
package foo.com;

import javax.wbem.cim.CIMException;
import javax.wbem.client.CIMOMHandle;
import javax.wbem.cim.CIMInstance;

/**
 * This Interface defines the methods that must be implemented by
 *  CIMBeanImpl and its subclasses. CIMBeanImpl constitutes the base
 *  class of the Java source generated by 'mofcomp -j'.
 */
public interface CIMBean {

    /**
     * This method returns the CIMBean's CIMOMHandle.
     *
     * @return    CIMOMHandle    handle to the CIMOM
     */
    public CIMOMHandle getCIMOMHandle();

    /**
     * This method sets the CIMBean's CIMOMHandle to the specifed value.
     *
     * @param    CIMOMHandle    handle to the CIMOM
     */
    public void setCIMOMHandle(CIMOMHandle handle);

    /**
     * This method returns the CIMBean's CIMInstance.
     *
     * @return    CIMInstance    handle to the CIMInstance being managed
     */
    public CIMInstance getCIMInstance();

    /**
     * This method sets the CIMBean's CIMInstance to the specified value.
     *
     * @param    CIMInstance    handle to the CIMInstance being managed
     */
    public void setCIMInstance(CIMInstance instance);

    /**
     * This method makes the remote call to update the CIMInstance in the
     * CIMOM.
     */
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
    public void update() throws CIMException;

    /**
     * This method makes the remote call to update the specified
     * CIMProperty of the CIMInstance in the CIMOM.
     *
     * @param    String    property name to update in the CIMInstance
     * @param    Object    property value to update in the CIMProperty
     */
    public void update(String propName, Object value) throws CIMException;

    /**
     * This method makes the remote call to delete the CIMInstance in the
     * CIMOM.
     */
    public void delete() throws CIMException;

    /**
     * This method returns a string array of the Key qualified property
     * name(s) in the CIMInstance. This is needed to build the
     * CIMObjectPath for the CIMInstance if it does not contain any
     * qualifier information.
     *
     * @return String Version qualifier value or "-1" if there isn't
     * one
     */
    public String[] getBeanKeys();

    /**
     * This method returns the CIM class's Version qualifier value or
     * '-1'if it does not have this qualifier.
     *
     * @return    String[]    array of the key qualified property names
     */
    public String getBeanVersion();

    /**
     * This method returns a string representation of the CIMBean.
     * This method is intended for debug purposes and the format of the
     * string may vary from implementation to implementation. The string
     * returned may be empty, but may not be null.
     *
     * @return    String    string representation of the Bean
     */
    public String toString();

} // Interface CIMBean
```

Following is the content of `CIMBeanImpl.java`:

```
package foo.com;

import java.io.Serializable;
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
import java.util.*;
import javax.wbem.client.CIMOMHandle;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.cim.CIMValue;
import javax.wbem.client.CIMOMHandle;

/**
 * This Class implements the CIMBean Interface. It is the base Class
 * of the Java source code generated by 'mofcomp -j'.
 */
public class CIMBeanImpl implements CIMBean, Serializable {

    private CIMInstance    cimInstance = null;
    private CIMOMHandle    cimomHandle = null;

    /**
     * This default constructor takes no parameters and creates an empty
     * instance of CIMBeanImpl.
     */
    public CIMBeanImpl() {

    super();

    } // constructor

    /**
     * This constructor takes the specified CIMOMHandle and CIMInstance and
     * creates a CIMBeanImpl.
     *
     * @param    CIMOMHandle    handle to the CIMOM
     * @param    CIMInstance    handle to the CIMInstance being managed
     */
    public CIMBeanImpl(CIMOMHandle handle, CIMInstance instance) {

    super();
    cimomHandle = handle;
    cimInstance = instance;

    } // constructor

    /**
     * This method returns the CIMBean's CIMOMHandle.
     *
     * @return    CIMOMHandle    handle to the CIMOM
     */
    public CIMOMHandle getCIMOMHandle() {

    return (cimomHandle);

    } // getCIMOMHandle
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
    /**
     * This method sets the CIMBean's CIMOMHandle to the specifed value.
     *
     * @param    CIMOMHandle    handle to the CIMOM
     */
    public void setCIMOMHandle(CIMOMHandle handle) {

    this.cimomHandle = handle;

    } // setCIMOMHandle

    /**
     * This method returns the CIMBean's CIMInstance.
     *
     * @return    CIMInstance    handle to the CIMInstance being managed
     */
    public CIMInstance getCIMInstance() {

    return (cimInstance);

    } // getCIMInstance

    /**
     * This method sets the CIMBean's CIMInstance to the specified
     * value.
     *
     * @param    CIMInstance    handle to the CIMInstance being managed
     */
    public void setCIMInstance(CIMInstance instance) {

    this.cimInstance = instance;

    } // setCIMInstance

    /**
     * This method makes the remote call to update the CIMInstance in
     * the CIMOM.
     */
    public void update() throws CIMException {

    cimomHandle.setInstance(getObjectPath(), cimInstance);

} // update

    /**
     * This method makes the remote call to update the specified
     * CIMProperty of the CIMInstance in the CIMOM.
     *
     * @param    String    property name to update in the CIMInstance
     * @param    Object    property value to update in the CIMProperty
     */
    public void update(String propName, Object value) throws CIMException {
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
cimomHandle.setProperty(getObjectPath(), propName, new CIMValue(value));

} // update

/**
 * This method makes the remote call to delete the CIMInstance in the
 * CIMOM.
 */
public void delete() throws CIMException {

cimomHandle.deleteInstance(getObjectPath());

} // delete

/**
 * This is a convenience method for use by subclasses to get the
 * Object contained in the given CIMProperty's CIMValue.
 * NOTE: The Object returned may be null.
 *
 * @param    String    property name whose value should be retrieved
 * @return    Object    object contained in the CIMProperty's CIMValue
 */
protected Object getProperty(String propName) {

try {

    return (cimInstance.getProperty(propName).getValue().getValue());

} catch (NullPointerException npe) {
}
return ((Object)null);

} // getProperty

/**
 * This is a convenience method for use by subclasses to get
 * the String[] equivalent to the Vector contained in the given
 * CIMProperty's CIMValue.
 * NOTE: The String[] returned may be null.
 *
 * @param    String    property name to get the value for
 * @param    String[]    property Values qualifier data
 * @param    Object[]    property ValueMap qualifier data
 * @return    String[]    container of constants for property value
 */
protected String[] getArrayProperty(String propName, String[]
valueArr,    Object[] valueMapArr) {

List propList = null;
try {

    propList =
    ((List)cimInstance.getProperty(propName).getValue().getValue());
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
} catch (NullPointerException npe) {
}

if (propList != null) {

    String[] returnArr;
    returnArr = new String[propList.size()];
    ListIterator listIterator = propList.listIterator();
    int counter = 0;
    while (listIterator.hasNext()) {

    returnArr[counter] = valueArr[getArrayIndex(valueMapArr,
        listIterator.next())];
    counter++;

    }
    return (returnArr);

}
return ((String[])null);

} // getArrayProperty

/**
 * This method gets the CIMInstance referenced by the property
 * value (i.e., the object path specified) and sets it in the
 * specified Bean. This method is used by accessor methods of
 * Association properties.
 *
 * @param    CIMObjectPath    object path for the CIMInstance
 * @param    CIMBeanImpl    Bean container for CIMInstance retrieved
 */
protected void getAssociationProperty(CIMObjectPath cop,
CIMBeanImp    bean) throws CIMException {

cop.setNameSpace("");
CIMInstance ci = cimomHandle.getInstance(cop, false, true, true,
    (String[])null);
bean.setCIMInstance(ci);
bean.setCIMOMHandle(cimomHandle);

} // getAssociationProperty

/**
 * This is a convenience method for use by subclasses to set a
 * CIMValue containing the specified Object value in the
 * CIMProperty of the specified name.
 *
 * @param    String    property name to set a new value for
 * @param    Object    property value to update in the CIMInstance
 */
protected void setProperty(String propName, Object propValue)throws
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
IllegalArgumentException {

cimInstance.setProperty(propName, new CIMValue(propValue));

} // setProperty

/**
 * This is a convenience method for use by subclasses to set a
 * CIMValue containing a Vector equivalent to the specified
 * String[] in the CIMProperty of the specified name.
 *
 * @param    String    property name to get the value for
 * @param    String[]    property Values qualifier data
 * @param    Object[]    property ValueMap qualifier data
 * @param    String[]    property value to set in the CIMInstance
 */
protected void setArrayProperty(String propName, String[] valueArr,
Object[] valueMapArr, String[] propValues) {

Vector vPropValue = new Vector(propValues.length);
for (int i = 0; i < propValues.length; i++) {

    vPropValue.addElement(valueMapArr[getArrayIndex(valueArr,
    propValues[i])]);

}
setProperty(propName, vPropValue);

} // setArrayProperty

/**
 * This method returns a string array of the Key qualified property
 * name(s) in the CIMInstance. This is needed to build the
 * CIMObjectPath for the CIMInstance if it does not contain any
 * qualifier information.
 *
 * @return    String[]    array of the key qualified property names
 */
public String[] getBeanKeys() {

return ((String[])null);

} // getBeanKeys

/**
 * This method returns the CIMObjectPath of the class's CIMInstance.
 *
 * @return    CIMObjectPath    object path for the CIMInstance
 */
protected CIMObjectPath getObjectPath() {

CIMObjectPath cop = new CIMObjectPath(cimInstance.getClassName());
Vector vKeys = cimInstance.getKeyValuePairs();
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
if ((vKeys != null) && (vKeys.size() > 0)) {

    cop.setKeys(vKeys);

} else {

    String[] keyArr = getBeanKeys();
    if (keyArr != null) {

    String keyProperty;
    for (int i = 0; i < keyArr.length; i++) {

        keyProperty = keyArr[i];
        cop.addKey(keyProperty,
        (cimInstance.getProperty(keyProperty)).getValue());

    }

    }

}
return (cop);

} // getObjectPath

/**
 * This convenience method returns the index of the specified
 * object in the specified array, or -1 if the object is not
 * contained in the array.
 *
 * @param    Object[]    Object array to find index of Object in
 * @param    Object    Object to find index of in Object array
 * @return    int    index of Object in Object array
 */
protected int getArrayIndex(Object[] objArr, Object obj) {

List arrList = Arrays.asList(objArr);
return (arrList.indexOf(obj));

} // getArrayIndex

/**
 * This method returns the CIM class's Version qualifier value, or
 * '-1'if it does not have this qualifier.
 *
 * @return    String    Version qualifier value or "-1" if there isn't
 * one
 */
public String getBeanVersion() {

return ("-1");

} // getBeanVersion
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
    /**
     * This method returns a string representation of the CIMBean.
     * This method is intended for debug purposes and the format of
     * the string may vary from implementation to implementation.
     * The string returned may be empty, but may not be null.
     *
     * @return    String    string representation of the Bean
     */
    public String toString() {

    return (cimInstance.toString());

    } // toString

} // Class CIMBeanImpl
```

Following is the content of `Simple_ClassBean.java`:

```
package foo.com;

import javax.wbem.client.*;
import javax.wbem.cim.*;
import java.util.*;
import java.lang.Exception;

/**
 * This Interface contains accessor and mutator methods for all
 * properties defined in CIM class Simple_Class as well as methods
 * comparable to the invokeMethods defined for this class. This Interface
 * is implemented by Simple_ClassBeanImpl. The CIM class Simple_Class is
 * described as follows:
 */
public interface Simple_ClassBean extends CIMBean {

    /**
     * This method returns the Simple_Class.Name property value. This
     * property is described as follows:
     *
     * Name of the class.
     *
     * @return    String    current Name property value
     * @exception    Exception
     */
    public String getName() throws Exception;

    /**
     * This method sets the Simple_Class.Name property value. This
     * property is described as follows:
     *
     * Name of the class.
     *
     * @param    String    new Name property value
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
     * @exception    Exception
     */
    public void setName(String name) throws Exception;


    /**
     * This method invokes the Simple_Class.printClass() method. This
     * method is described as follows:
     *
     * Method to print the contents of the class.
     *
     * @return    String    return value of printClass() invokation
     * @exception    Exception
     */
    public String printClass() throws Exception, CIMException;

} // Interface Simple_ClassBean
```

Following is the content of `Simple_ClassBeanImpl.java`:

```
package foo.com;

import javax.wbem.client.*;
import javax.wbem.cim.*;
import java.util.*;
import java.lang.Exception;

/**
 * This Class contains accessor and mutator methods for all properties
 * defined in the CIM class Simple_Class as well as methods comparable
 * to the invokeMethods defined for this class. This Class implements
 * the Simple_ClassBean Interface. The CIM class Simple_Class is
 * described as follows:
 */
public class Simple_ClassBeanImpl extends CIMBeanImpl implements
    Simple_ClassBean {

     private CIMOMHandle cimomHandle = null;
    private CIMInstance cimInstance = null;
    private final static String[] keysArr = {"Name"};

    /**
     * This constructor creates a Simple_ClassBeanImpl Class which
     * implements the Simple_ClassBean Interface, and encapsulates the
     * CIM class Simple_Class in a Java Bean. The CIM class Simple_Class
     * is described as follows:
     *
     * @param    CIMOMHandle    handle to the CIMOM
     * @param    CIMInstance    handle to the CIMInstance being managed
     */
    public Simple_ClassBeanImpl(CIMOMHandle handle, CIMInstance instance)
    {
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
 super(handle, instance);

this.cimomHandle = handle;
this.cimInstance = instance;

} // constructor

/**
 * This method returns an array of Strings with the names of the key
 * qualified properties defined for the CIM class. This method is
 * used to build the CIMObjectPath of the CIMInstance managed by
 * the Bean in the case that the key qualifiers are not included
 * in the CIMInstance.
 *
 * @return    String[]    array of the key qualified property names
 */
public String[] getBeanKeys() {

return keysArr;

} // getBeanKeys

/**
 * This method returns the Simple_Class.Name property value. This
 * property is described as follows:
 *
 * Name of the class.
 *
 * @return    String    current Name property value
 * @exception    Exception
 */
public String getName() throws Exception {

return (String)getProperty("Name");

} // getName

/**
 * This method sets the Simple_Class.Name property value. This
 * property is described as follows:
 *
 * Name of the class.
 *
 * @param    String    new Name property value
 * @exception    Exception
 */
public void setName(String name) throws Exception {

setProperty("Name", name);

} // setName
```

EXAMPLE 2–1 Generating JavaBeans *(Continued)*

```
    /**
     * This method invokes the Simple_Class.printClass() method. This
     * method is described as follows:
     *
     * Method to print the contents of the class.
     *
     * @return    String    return value of printClass() invokation
     * @exception    Exception
     */
    public String printClass() throws Exception, CIMException {

    Vector vInParams = new Vector();
    Vector vOutParams = new Vector();

    CIMValue cv = cimomHandle.invokeMethod(cimInstance.getObjectPath(),
 "printClass", vInParams, vOutParams);
    return (String)cv.getValue();

    } // printClass
} // Class Simple_ClassBeanImpl
```

CHAPTER 3

Writing a Client Program

This chapter explains how to use the Solaris WBEM SDK client APIs (`javax.wbem.client`) to write client programs, and includes the following topics:

- "Overview" on page 45
- "Opening and Closing a Client Connection" on page 46
- "Performing Basic Client Operations" on page 48
- "Setting Access Control" on page 68
- "Working with Qualifiers and Qualifier Types" on page 71
- "Batching Client Requests" on page 72
- "Handling CIM Events" on page 74
- "Reading and Writing Log Messages" on page 82

Note – For detailed information on the WBEM client APIs (`javax.wbem.client`), see file:/usr/sadm/lib/wbem/doc/index.html.

Overview

WBEM client applications use the `javax.wbem.client` APIs to manipulate Common Information Model (CIM) objects. A client application uses the CIM API to construct an object (for example, a class, instance, or namespace) and then initializes, or instantiates, that object. The application uses the client APIs to pass the object to the CIM Object Manager and request a WBEM operation, such as creating a CIM class, instance, or namespace.

Sequence of a Client Application

Client applications typically follow this sequence:

1. Connect to the CIM Object Manager using `CIMClient`. A client application connects to the CIM Object Manager each time it needs to perform a WBEM operation, such as creating a CIM class or updating a CIM instance. See "Opening and Closing a Client Connection" on page 46.
2. Use the client APIs to request operations and perform programming tasks. The application's feature set determines which operations it needs to request. The tasks that most programs perform include creating, deleting and updating instances; enumerating objects; calling methods; retrieving class definitions; and handling errors. Client programs can also create and delete classes, and create and delete namespaces, and use qualifiers. See "Performing Basic Client Operations" on page 48.
3. Close the client connection to the CIM Object Manager using `CIMClient`, to free the server resources used by the client session. See "Opening and Closing a Client Connection" on page 46.

Opening and Closing a Client Connection

A client application must first establish a connection with the CIM Object Manager before it can perform WBEM operations such as adding, modifying, or deleting a CIM class, CIM instance, or CIM qualifier type. The client application and CIM Object Manager can run on the same host or on different hosts. In addition, multiple clients can establish connections to the same CIM Object Manager.

About Namespaces

When an application connects to the CIM Object Manager, it must also connect to a *namespace*, where all subsequent operations occur. A namespace is a directory-like structure that contains classes, instances, and qualifier types. The names of all objects within a namespace must be unique. When you install the Solaris WBEM SDK, four namespaces are created:

- `root\cimv2` – The default namespace. Contains the CIM classes that represent objects on the system on which Solaris WBEM Services is installed.
- `root\security` – Contains the security classes.
- `root\snmp`– Contains the SNMP adapter classes.
- `root\system`– Contains the classes that manage the CIM Object Manager.

Opening a Client Connection

To open a client connection, you use the `CIMClient` class to connect to the CIM Object Manager. The `CIMClient` class takes four arguments:

- *name* – Required. An instance of a `CIMNameSpace` object that contains the name of the host and the namespace used for the client connection. The default is `root\cimv2` on the local host (the local host is the same host in which the client application is running). Once connected to the CIM Object Manager, all subsequent `CIMClient` operations occur within the specified namespace.
- *principal* – Required. An instance of a `UserPrincipal` object that contains the name of a valid Solaris user account. The CIM Object Manager checks the access privileges for the user name to determine the type of access allowed to CIM objects.
- *credential* – Required. An instance of a `PasswordCredential` object that contains a valid password for the `UserPrincipal` Solaris account.
- *protocol* – Optional (string). Protocol used for sending messages to the CIM Object Manager; either *RMI* (the default), or *HTTP*.

EXAMPLE 3–1 Connecting to the Root Account

In this example, the application connects to the CIM Object Manager running on the local host in the default namespace. The application creates a `UserPrincipal` object for the root account, which has read and write access to all CIM objects in the default namespace.

```
{
    ...

   /* Create a namespace object initialized with two null strings
   that specify the default host (the local host) and the default
   namespace (root\cimv2).*/

    CIMNameSpace cns = new CIMNameSpace("", "");

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");
    /* Connect to the namespace as root with the root password. */

    CIMClient cc = new CIMClient(cns, up, pc);
    ...
}
```

EXAMPLE 3–2 Connecting to a User Account

In this example, the application first creates an instance of a `CIMNameSpace`, `UserPrincipal`, and `PasswordCredential` object. Then, the application uses the `CIMClient` class to connect to the CIM Object Manager and pass the host name, namespace , user name, and password credential to the CIM Object Manager.

```
{
    ...
```

EXAMPLE 3–2 Connecting to a User Account *(Continued)*

```
    /* Create a namespace object initialized with A
    (name of namespace) on host happy.*/
    CIMNameSpace cns = new CIMNameSpace("happy", "A");
    UserPrincipal up = new UserPrincipal("Mary");
    PasswordCredential pc = new PasswordCredential("marys_password");
    CIMClient cc = new CIMClient(cns, up, pc);
    ...
    ...
}
```

EXAMPLE 3–3 Authenticating as an RBAC Role Identity

You use the `SolarisUserPrincipal` and `SolarisPasswordCredential` classes to authenticate a user's role identity. This example authenticates as `Mary` and assumes the role `Admin`.

```
{
...
CIMNameSpaceRole cns = new CIMNameSpace("happy", "A");
SolarisUserPrincipal sup = new SolarisUserRolePrincipal("Mary", "Admin");
SolarisPswdCredential spc = new
        SolarisPswdCredential("marys_password", "admins_password");
CIMClient cc = new CIMClient(cns, sup, spc);
```

Closing a Client Connection

Use the `close` method of the `CIMClient` class to close a client connection and free the server resources used by the session.

EXAMPLE 3–4 Closing a Client Connection

This example closes a client connection. The instance variable *cc* represents the client connection.

```
...
cc.close();
...
```

Performing Basic Client Operations

This section describes how to use the the `javax.wbem.client` APIs to request operations and perform common programming tasks.

Creating an Instance

Use the `newInstance` method to create an instance of an existing class. If the existing class has a key property, the application must set the key property to a unique value. As an option, an instance can define additional qualifiers that are not defined for the class. These qualifiers can be defined for the instance or for a particular property of the instance and do not need to appear in the class declaration.

Applications can use the `getQualifiers` method to get the set of qualifiers defined for a class.

EXAMPLE 3–5 Creating an Instance

This example uses the `newInstance` method to create a Java class representing a CIM instance, for example, a Solaris package, from the `Solaris_Package` class.

```
...
{
/*Connect to the CIM Object Manager in the root\cimv2
namespace on the local host. Specify the username and password of an
account that has write permission to the objects in the
root\cimv2 namespace. */

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");
    /* Connect to the namespace as root with the root password. */

    CIMClient cc = new CIMClient(cns, up, pc);
...

// Get the Solaris_Package class
cimclass = cc.getClass(new CIMObjectPath("Solaris_Package"),
                                          true, true, true, null);

/* Create a new instance of the Solaris_Package
 class populated with the default values for properties. If the provider
 for the class does not specify default values, the values of the
 properties will be null and must be explicitly set. */

 CIMInstance ci = cc.createInstance (new CIMObjectPath("Solaris_Package"),
 ci);
}
...
```

Deleting an Instance

Use the `deleteInstance` method to delete an instance.

EXAMPLE 3–6 Deleting Instances

The following example connects the client application to the CIM Object Manager, uses `CIMObjectPath` to construct an object containing the CIM object path of the object to be deleted, `enumerateInstance` to get the instance and all instances of its subclasses, and `deleteInstance` to delete each instance.

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.CIMClient;
import javax.wbem.client.PasswordCredential;
import javax.wbem.client.UserPrincipal;

/**
 * Returns all instances of the specified class.
 * This example takes five arguments: hostname (args[0]), username
 * (args[1]), password (args[2]) namespace (args[3] and classname (args[4])
 * It will delete all instances of the specified classname.  The specified
 * username must have write permissions to the specified namespace
 */
public class DeleteInstances {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        // if not five arguments, show usage and exit
        if (args.length != 5) {
           System.out.println("Usage: DeleteInstances host username " +
                              "password namespace classname ");
           System.exit(1);
           }
       try {
          // args[0] contains the hostname and args[3] contains the
          // namespace.  We create a CIMNameSpace (cns) pointing to
          // the specified namespace on the specified host
          CIMNameSpace cns = new CIMNameSpace(args[0], args[3]);

         // args[1] and args[2] contain the username and password.
         // We create a UserPrincipal (up) using the username and
         // a PasswordCredential using the password.
         UserPrincipal up = new UserPrincipal(args[1]);
         PasswordCredential pc = new PasswordCredential(args[2]);

        // Connect to the CIM Object Manager and pass it the
        // CIMNameSpace, UserPrincipal and PasswordCredential objects
        // we created.
        cc = new CIMClient(cns, up, pc);

        // Get the class name (args[4]) and create a CIMObjectPath
```

EXAMPLE 3–6 Deleting Instances *(Continued)*

```
        CIMObjectPath cop = new CIMObjectPath(args[4]);

        // Get an enumeration of all the instance object paths of the
        // class and all subclasses of the class.  An instance object
        // path is a reference used by the CIM object manager to
        // locate the instance
        Enumeration e = cc.enumerateInstanceNames(cop);


        // Iterate through the instance object paths in the enumeration.
        // Construct an object to store the object path of each
        // enumerated instance, print the instance and then delete it
        while (e.hasMoreElements()) {
           CIMObjectPath  op = (CIMObjectPath)e.nextElement();
           System.out.println(op);
           cc.deleteInstance(op);
        } // end while
  } catch (Exception e) {
       // is we have an exception, catch it and print it out.
       System.out.println("Exception: "+e);
  } // end catch

 // close session.
 if (cc != null) {
     cc.close();
       }
    }
}
```

Getting and Setting Instances

Client applications commonly use the `getInstance` method to retrieve CIM instances from the CIM Object Manager. When an instance of a class is created, it inherits the properties of the class it is derived from and all parent classes in its class hierarchy. The `getInstance` method takes the Boolean argument *localOnly*.

- If *localOnly* is true, `getInstance` returns only the non-inherited properties in the specified instance. The non-inherited properties are those defined in the instance itself.
- If *localOnly* is false, all properties in the class are returned; those defined in the instance, and all properties inherited from all parent classes in its class hierarchy.

Use the `setInstance` method to update an existing instance.

EXAMPLE 3–7 Getting and Setting Instances

The following example gets instances of an object path in an enumeration, updates the property value of b to 10 in each instance, and passes the updated instances to the CIM Object Manager.

EXAMPLE 3–7 Getting and Setting Instances *(Continued)*

```
...
{
    // Create an object path, an object that contains the CIM name for
    // "myclass"
    CIMObjectPath cop = new CIMObjectPath("myclass");

    /* Get instances for each instance object path in an enumeration,
    update the property value of b to 10 in each instance, and pass the
    updated instance to the CIM Object Manager. */

    while(e.hasMoreElements()) {
           CIMInstance ci = cc.getInstance((CIMObjectPath)(e.nextElement
                                        ()),true, true, true, null);
           ci.setProperty("b", new CIMValue(new Integer(10)));
           cc.setInstance(new CIMObjectPath(),ci);
           }
}
...
```

Getting and Setting Properties

A CIM property is a value that describes the characteristic of a CIM class. Properties can be thought of as a pair of functions, one that *gets* the property value and one that *sets* the property value.

EXAMPLE 3–8 Getting a Property

The following example uses `enumerateInstanceNames` to return the names of all instances of the Solaris processor, `getProperty` to get the value of the current clockspeed for each instance, and `println` to print the current clockspeed values.

```
...
{
/* Create an object (CIMObjectPath) to store the name of the
Solaris_Processor class. */

CIMObjectPath cop = new CIMObjectPath("Solaris_Processor");

/* The CIM Object Manager returns an enumeration containing the names
of instances of the Solaris_Processor class. */

Enumeration e = cc.enumerateInstanceNames(cop);

/* Iterate through the enumeration of instance object paths.
Use the getProperty method to get the current clockspeed
value for each Solaris processor. */

while(e.hasMoreElements()) {
        CIMValue cv = cc.getProperty(e.nextElement(CIMObjectPath),
```

EXAMPLE 3–8 Getting a Property *(Continued)*

```
                                        "CurrentClockSpeed");
        System.out.println(cv);
}
...
}
```

EXAMPLE 3–9 Setting a Property

The following example sets the initial shell value for all `Solaris_UserTemplate` instances. This code segment uses `enumerateInstanceNames` to get the names of all instances of the Solaris_User Template, and `setProperty` to set the value of the initial shell for each instance.

```
...
{
    /* Create an object (CIMObjectPath) to store the name of the
    Solaris_Processor class. */

    CIMObjectPath cop = new CIMObjectPath("Solaris_UserTemplate");

    /* The CIM Object Manager returns an enumeration containing the names
    of instances of the Solaris_UserTemplate class and
    all its subclasses. */

    Enumeration e = cc.enumerateInstanceNames(cop);

    /* Iterate through the enumeration of instance object paths.
    Use the setProperty method to set the initial shell
    value to /usr/bin/sh for each Solaris_UserTemplate instance. */

   for (; e.hasMoreElements(); cc.setProperty(e.nextElement(),
        "/usr/bin/sh", new CIMValue(new Integer(500))));

...
}
```

Enumerating Objects

An *enumeration* is a collection of objects that can be retrieved one at a time. You can enumerate classes, class names, instances, instance names, and namespaces. The results of an enumeration depend on the method and the arguments used, as shown in the following table.

Enumerating Objects

TABLE 3–1 Enumerating Objects

Method	No args	*deep*	*localOnly*
`enumerateClasses`	Returns the contents of the class specified in *path*.	**If true:** Returns the contents of the subclasses of the specified class, but does not return the class itself.	**If true:** Returns only non-inherited properties and methods of the specified class.
		If false: Returns the contents of the direct subclasses of the specified class.	**If false:** Returns all properties of the specified class.
`enumerateInstances`	Returns the instances of the class specified in *path*.	**If true:** Returns the instances of the specified class and its subclasseses.	**If true:** Returns only non-inherited properties of the instances of the specified class.
		If false: Returns the instances of the specified class and its subclasses. The properties of the subclasses are filtered out.	**If false:** Returns all properties of the instances of the specified class.
enumerateClassNames	Returns the names of the class specified in *path*.	**If true:** Returns the names of all classes derived from the specified class.	N/A
		If false: Returns only the names of the first level children of the specified class.	N/A
enumerateInstanceNames	Returns the names of the instances of the class specified in *path*.	N/A	N/A
		N/A	N/A
enumNameSpace	Returns a list of the namespaces within the namespace specified in *path*	**If true:** Returns the entire hierarchy of namespaces under the specified namespace.	N/A
		If false: Returns only the first level children of the specified namespace.	N/A

EXAMPLE 3–10 Enumerating Classes

The following example program returns the contents of a class and its subclasses.

EXAMPLE 3–10 Enumerating Classes *(Continued)*

```
...
{
    /* Creates a CIMObjectPath object and initializes it
    with the name of the CIM class to be enumerated (myclass). */

    CIMObjectPath cop = new CIMObjectPath(myclass);

    /* This enumeration contains the classes and subclasses
    in the enumerated class (deep=true). This enumeration
    returns only the non-inherited methods and properties
    for each class and subclass (localOnly is true).*/

    Enumeration e = cc.enumerateClasses(cop, true, true);
}
...
```

EXAMPLE 3–11 Enumerating Classes and Instances

The following example program performs a deep and shallow (*deep=false*) enumeration of classes and instances. The *localOnly* flag returns the contents of the classes and instances instead of the names of the classes and instances.

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.client.CIMClient;
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;


/**
 * This example enumerates classes and instances. It does deep and
 * shallow enumerations on a class that is passed from the command line
 */
public class ClientEnum {

    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        CIMObjectPath cop = null;
        if (args.length < 4) {
            System.out.println("Usage: ClientEnum host user passwd " +
                               "classname");
            System.exit(1);
   }
   try {
       CIMNameSpace cns = new CIMNameSpace(args[0]);
```

EXAMPLE 3–11 Enumerating Classes and Instances *(Continued)*

```
UserPrincipal up = new UserPrincipal(args[1]);
PasswordCredential pc = new PasswordCredential(args[2]);
cc = new CIMClient(cns, up, pc);


// Get the class name from the command line
cop = new CIMObjectPath(args[3]);
// Do a deep enumeration of the class
Enumeration e = cc.enumerateClasses(cop, true, true, true,
                                    true);
// Will print out all the subclasses of the class.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
// Do a shallow enumeration of the class
e = cc.enumerateClasses(cop, false, true, true, true);
// Will print out the first level subclasses.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
// Do a deep enumeration of the instances of the class
e = cc.enumerateInstances(cop, false, true, true, true, null);
// Will print out all the instances of the class and its
// subclasses.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
// Do a shallow enumeration of the instances of the class
e = cc.enumerateInstances(cop, false, false, true, true, null);
// Will print out all the instances of the class.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");

e = cc.enumerateInstanceNames(cop);
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");

e = cc.enumerateInstanceNames(cop);
while (e.hasMoreElements()) {
    CIMObjectPath opInstance = (CIMObjectPath)e.nextElement();
    CIMInstance ci = cc.getInstance(opInstance, false,
                                    true, true, null);
    System.out.println(ci);
}
System.out.println("+++++");
```

EXAMPLE 3–11 Enumerating Classes and Instances *(Continued)*

```
  }
  catch (Exception e) {
      System.out.println("Exception: "+e);
  }

 // close session.
 if (cc != null) {
     cc.close();
  }
 }
}
```

EXAMPLE 3–12 Enumerating Class Names

The following example program returns a list of class names and subclass names.

```
...
{
    /* Creates a CIMObjectPath object and initializes it
    with the name of the CIM class to be enumerated (myclass). */
    CIMObjectPath cop = new CIMObjectPath(myclass);

    /* This enumeration contains the names of the classes and subclasses
    in the enumerated class. */
    Enumeration e = cc.enumerateClassNames(cop, true);
}
...
```

EXAMPLE 3–13 Enumerating Namespaces

This example program uses the `enumNameSpace` method in the `CIMClient` class to print the names of the namespace and all the namespaces contained within the namespace.

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.CIMClient;
import javax.wbem.client.PasswordCredential;
import javax.wbem.client.UserPrincipal;


/**
 *
```

EXAMPLE 3–13 Enumerating Namespaces *(Continued)*

```
 */
public class EnumNameSpace {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        // if not four arguments, show usage and exit
        if (args.length < 4) {
            System.out.println("Usage: EnumNameSpace host username " +
                               "password namespace");
            System.exit(1);
            }
        try {
            // args[0] contains the hostname. We create a CIMNameSpace
            // (cns) pointing to the specified namespace on the
            // specified host
            CIMNameSpace cns = new CIMNameSpace(args[0], "");

           // args[1] and args[2] contain the username and password.
           // We create a UserPrincipal (up) using the username and
           // a PasswordCredential using the password.
           UserPrincipal up = new UserPrincipal(args[1]);
           PasswordCredential pc = new PasswordCredential(args[2]);

           // Connect to the CIM Object Manager and pass it the
           // CIMNameSpace, UserPrincipal and PasswordCredential objects
           // we created.
           cc = new CIMClient(cns, up, pc);

           // Use the namespace (args[3]) to create a CIMObjectPath
           CIMObjectPath cop = new CIMObjectPath("", args[3]);

           // Enumerate the namespace
           Enumeration e = cc.enumNameSpace(cop);
           while (e.hasMoreElements()) {
               System.out.println((CIMObjectPath)e.nextElement());
           } // end while

           } catch (Exception e) {
              // is we have an exception, catch it and print it out.
              System.out.println("Exception: "+ e);
           } // end catch

          // close session.
          if (cc != null) {
              cc.close();
        }
    }
}
```

EXAMPLE 3–13 Enumerating Namespaces *(Continued)*

Creating Associations

An *association* describes a relationship between two or more managed resources such as a computer and its hard disk. This relationship is abstracted in an *association class*, which is a special type of class that contains an *association qualifier*. You can add or change an association class without affecting the objects themselves.

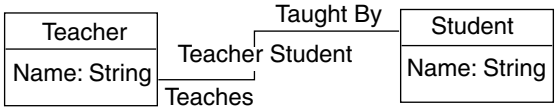

FIGURE 3–1 TeacherStudent Association 1

Figure 3–1 shows two classes, `Teacher` and `Student`. Both classes are linked by the `TeacherStudent` association. The `TeacherStudent` association has two references:

- `Teaches`, a property that refers to an instance of the `Teacher` class
- `TaughtBy`, a property that refers to an instance of the `Student` class

About The Association Methods

The association methods in `CIMClient` return information about the relationships between classes and instances. These methods are described in the following table.

TABLE 3–2 Association Methods

Method	Description
`associators`	Gets the CIM classes or instances that are associated with the specified CIM class or instance.
`associatorNames`	Gets the names of the CIM classes or instances that are associated with the specified CIM class or instance.
`references`	Gets the association classes or instances that refer to the specified CIM class or instance, respectively.
`referenceNames`	Gets the names of the association classes or instances that refer to the specified CIM classes or instances, respectively.

These methods take one required argument, `CIMObjectPath`, which is the name of a source CIM class or CIM instance whose associations, associated classes, or instances you want to return. If the CIM Object Manager does not find any associations, associated classes, or instances, it does not return anything.

- If `CIMObjectpath` is a class, the methods return the associated classes and the subclasses of each associated class.
- If `CIMObjectpath` is an instance, the methods return the instances of the associated class and the subclasses of each associated class.

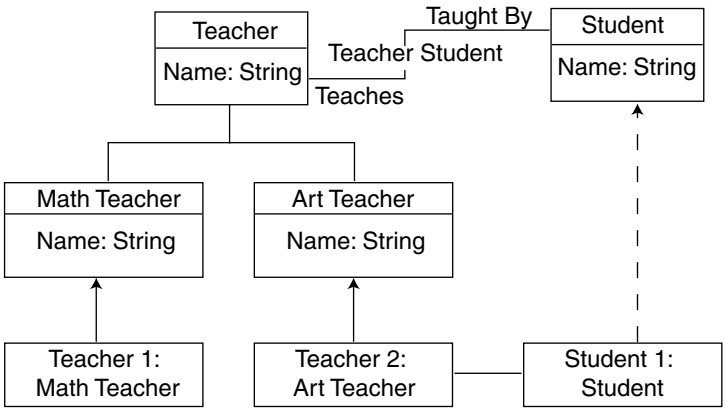

FIGURE 3–2 TeacherStudent Association 2

In Figure 3–2, the `associators` and `associatorNames` methods return information about the classes associated with the `Teacher` and `Student` classes. The `references` and `referenceNames` methods return information about the associations between the `Teacher` and `Student` classes.

TABLE 3–3 TeacherStudent Methods

Example	Output	Description
`associators(Teacher, null, null, null, null, false, false, null)`	`Student` class	Returns associated classes.`Student` is linked to `Teacher` by the `TeacherStudent` association.
`associators(MathTeacher, null, null, null, null,,false, false, null)`	`Student`	Returns associated classes. `Teacher` is linked to `Student` by the `TeacherStudent` association. `MathTeacher` and `ArtTeacher` inherit the `TeacherStudent` association from `Teacher`.
`associatorNames(Teacher, null, null, null, null)`	Name of the `Student` class	Returns the names of the associated classes. `Student` is linked to `Teacher` by the `TeacherStudent`association.

TABLE 3–3 TeacherStudent Methods *(Continued)*

Example	Output	Description
`references(Student, null, null. false, false, null)`	`TeacherStudent`	Returns the associations in which `Student` participates.
`references(Teacher, null, null. false, false, null)`	`TeacherStudent`	Returns the associations in which `Teacher` participates.
`references(Teacher, null, null, false, false, null)`	`TeacherStudent`	Returns the associations in which `Teacher` participates.
`referenceNames(Teacher, null, null)`	The name of the `TeacherStudent` class.	Returns the names of the associations in which `Teacher` participates.
`referenceNames(Teacher, null, null)`	The name of the `TeacherStudent` class.	Returns the names of the associations in which `Teacher` participates.

Note – The `associatorNames` and `referenceNames` methods do not take the arguments *includeQualifiers*, *includeClassOrigin*, and *propertyList* because these arguments are irrelevant to a method that returns only the names of instances or classes, not their entire contents.

Passing a Class to the Association Methods

To specify the name of a class, you specify its *model path*. The model path includes the class's namespace, class name, and keys. A *key* is a property or set of properties that uniquely identify managed resource. Key properties are marked with the `key` qualifier. This model path:

```
\\myserver\\root\cimv2\Solaris_ComputerSystem.Name=
mycomputer: CreationClassName=Solaris_ComputerSystem
```

Specifies the following:

- `\\myserver\root\cimv2` is the default CIM namespace on host `myserver`.
- `Solaris_ComputerSystem` is the name of the class from which the instance is derived.
- `Name=mycomputer, CreationClassName=Solaris_ComputerSystem` are two key properties in the format `key property=value`.

Passing Instances to the Association Methods

You use the `enumerateInstances` method to return all instances of a given class, and a loop structure to iterate through the instances. In the loop, you can pass each instance to an association method.

EXAMPLE 3–14 Passing Instances

This example enumerates the instances in the `op` class and its subclasses, uses a `while` loop to cast each instance to a `CIMObjectPath` (`op`), and passes each instance as the first argument to the `associators` method.

```
{
    ...
    Enumeration e = cc.enumerateInstances(op, true);
    while (e.hasMoreElements()) {
        op = (CIMObjectPath)e.nextElement();
        Enumeration e1 = cc.associators(op, null, null,
                null, null, false, false, null);
    ...
    }
```

Using Optional Arguments with the Association Methods

You can use the optional arguments with the association methods to filter the classes and instances that are returned. Each optional parameter value passes its results to the next parameter for filtering until all parameters have been processed.

You can pass values for any one or a combination of the optional parameters. You must enter a value or null for each parameter. The `assocClass`, `resultClass`, `role`, and `resultRole` parameters filter the classes and instances that are returned. Only the classes and instances that match the values specified for these parameters are returned. The `includeQualifiers`, `includeClassOrigin`, and `propertyList` parameters filter the information that are included in the classes and instances that are returned.

Calling Methods

You use the `invokeMethod` interface to call a method in a class supported by a provider. To retrieve the signature of a method, an application must first get the definition of the class to which the method belongs. The `invokeMethod` method returns a `CIMValue`. The return value is null when the method you invoke does not define a return value.

The `invokeMethod` interface takes four arguments, as described in the following table.

TABLE 3–4 `invokeMethod` Parameters

Parameter	Data Type	Description
name	CIMObjectPath	The name of the instance on which the method must be invoked

TABLE 3–4 `invokeMethod` Parameters *(Continued)*

Parameter	Data Type	Description
methodName	String	The name of the method to call
inParams	Vector	Input parameters to pass to the method
outParams	Vector	Output parameters to get from the method

EXAMPLE 3–15 Calling a Method

This example gets the instances of the `CIM_Service` class (services that manage device or software features) and uses the `invokeMethod` method to stop each service.

```
{
    ...
    /* Pass the CIM Object Path of the CIM_Service class
    to the CIM Object Manager. We want to invoke a method defined in
    this class. */

    CIMObjectPath op = new CIMObjectPath("CIM_Service");

    /* The CIM Object Manager returns an enumeration of instance
    object paths, the names of instances of the CIM_Service
    class. */

    Enumeration e = cc.enumerateInstanceNames (op, true);

    /* Iterate through the enumeration of instance object paths */

    while(e.hasMoreElements()) {
                // Get the instance
                CIMObjectPath op = (CIMObjectPath) e.nextElement();
                //Invoke the Stop Service method to stop the CIM services.
                cc.invokeMethod("StopService", null, null);
              }
}
```

Retrieving Class Definitions

The `getClass` method gets a CIM class. When a class is created, it inherits the methods and properties of the class from which it is derived, and all parent classes in the class hierarchy. The `getClass` method takes the *localOnly* Boolean argument.

- If *localOnly* is true, `getClass` returns only non-inherited properties and methods.
- If *localOnly* is false, `getClass` returns all properties in the class.

EXAMPLE 3–16 Retrieving a Class Definition

This example uses the following methods to retrieve a class definition:

- `CIMNameSpace` – Create a new namespace

EXAMPLE 3–16 Retrieving a Class Definition *(Continued)*

- `CIMClient` – Create a new client connection to the CIM Object Manager
- `CIMObjectPath` – Create an object path, which is an object to contain the name of the class to retrieve
- `getClass` – Retrieve the class from the CIM Object Manager

```
import java.rmi.*;
import javax.wbem.client.CIMClient;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMValue;
import javax.wbem.cim.CIMProperty;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import java.util.Enumeration;
/**
 * Gets the class specified in the command line. Works in the default
 * namespace root\cimv2.
 */
public class GetClass {
    public static void main(String args[]) throws CIMException {
      CIMClient cc = null;
      try {
         CIMNameSpace cns = new CIMNameSpace(args[0]);
         UserPrincipal up = new UserPrincipal("root");
         PasswordCredential pc = new PasswordCredential("root_password");
         cc = new CIMClient(cns);
         CIMObjectPath cop = new CIMObjectPath(args[1]);
             // Returns only the methods and properties that
             // are local to the specified class (localOnly is true).
         cc.getClass(cop, true);
      } catch (Exception e) {
         System.out.println("Exception: "+e);
      }
      if(cc != null) {
         cc.close();
        }
    }
}
```

Handling Exceptions

Each `CIMClient` method throws a `CIMException`, or error condition. The CIM Object Manager uses Java exception handling and creates a hierarchy of WBEM-specific exceptions. The `CIMException` class is the base class for CIM exceptions. All other CIM exception classes extend from the `CIMException` class.

Each class of CIM exceptions defines a particular type of error condition that the API code handles. `CIMException` has methods to retrieve error codes and parameters relating to the exception. Refer to file:/usr/sadm/lib/wbem/doc/index.html for more information on the `CIMException` class.

Creating a Namespace

The Solaris WBEM SDK installation compiles the standard CIM Managed Object Format (MOF) files into the default namespaces. If you create a new namespace, you must compile the appropriate CIM `.mof` files into the new namespace before you create objects in that namespace. For example, if you plan to create classes that use the standard CIM elements, compile the CIM Core Schema into the namespace. If you plan to create classes that extend the CIM Application Schema, compile the CIM Application into the namespace.

EXAMPLE 3–17 Creating a Namespace

This example uses a two-step process to create a namespace within an existing namespace:

1. When the namespace is created, the `CIMNameSpace` method constructs a namespace object that contains the parameters to be passed to the CIM Object Manager.
2. The `CIMClient` class connects to the CIM Object Manager and passes the namespace object. The CIM Object Manager creates the namespace, using the parameters contained in the namespace object.

```
{
    ...
    /* Creates a namespace object on the client, which stores parameters
    passed to it from the command line. args[0] contains the host
    name (for example, myhost); args[1] contains the
    parent namespace (for example, the toplevel directory.) */

    CIMNameSpace cns = new CIMNameSpace (args[0], args[1]);

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");

    /* Connects to the CIM Object Manager and passes it three parameters:
    the namespace object (cns), which contains the host name (args[0]) and
    parent namespace name (args[1]), a user name string (args[3]), and a
    password string (args[4]). */

    CIMClient cc = new CIMClient (cns, up, pc);

    /* Passes to the CIM Object Manager another namespace object that
    contains a null string (host name) and args[2], the name of a
    child namespace (for example, secondlevel). */
```

EXAMPLE 3–17 Creating a Namespace *(Continued)*

```
    CIMNameSpace cop = new CIMNameSpace("", args[2]);

    /* Creates a new namespace by the name passed in as args[2] under the
    toplevel namespace on myhost./*

    cc.createNameSpace(cop);
    ...
}
```

Deleting a Namespace

Use the `deleteNameSpace` method to delete a namespace.

Creating a Base Class

Note – You can also create a base class using the MOF language. If you are familiar with MOF syntax, use a text editor to create a MOF file and then use the MOF Compiler to compile the file into Java classes. See Chapter 2.

Use the `CIMClass` class to create a Java class representing a CIM class. To declare the most basic class, you need only specify the class name and a key property or an abstract qualifier. However, most classes include properties that describe the data of the class. To declare a property, include the property's data type, name, and an optional default value. The property data type must be an instance of `CIMDataType`.

A property can have a key qualifier, which identifies it as a *key property*. A key property uniquely defines the instances of the class. Only keyed classes can have instances. Therefore, if you do not define a key property in a class, the class can only be used as an abstract class. If you define a key property in a class in a new namespace, you must first compile the core MOF files into the namespace. The core MOF files contain the declarations of the standard CIM qualifiers, such as the *key* qualifier.

Class definitions can be more complicated, including such features as aliases, qualifiers, and qualifier flavors.

Deleting a Class

Use the `CIMClient deleteClass` method to delete a class. Deleting a class removes the class and throws a `CIMException`.

Note – You must first remove any existing subclasses or instances before deleting a base class.

EXAMPLE 3–18 Deleting a Class

This example uses the `deleteClass` method to delete a class in the default namespace `root\cimv2`. This program takes four required string arguments (*hostname*, *classname*, *username*, and *password*). The user running this program must specify the username and password for an account that has write permission to the `root\cimv2` namespace.

```
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.client.CIMClient;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;

import java.rmi.*;
import java.util.Enumeration;

/**
 * Deletes the class specified in the command line. Works in the default
 * namespace root/cimv2.
 */
public class DeleteClass {
    public static void main(String args[]) throws CIMException {
       CIMClient cc = null;
       // if not four arguments, show usage and exit
       if (args.length != 4) {
           System.out.println("Usage: DeleteClass host className " +
                              "username password");
           System.exit(1);
       }
       try {
          // args[0] contains the hostname. We create a CIMNameSpace
          // (cns) pointing to the default namespace on the specified host
          CIMNameSpace cns = new CIMNameSpace(args[0]);

          // args[2] and args[3] contain the username and password.
          // We create a UserPrincipal (up) using the username and
          // a PasswordCredential using the password.
          UserPrincipal up = new UserPrincipal(args[2]);
          PasswordCredential pc = new PasswordCredential(args[3]);

          cc = new CIMClient(cns, up, pc);

          // Get the class name (args[4]) and create a CIMObjectPath
          CIMObjectPath cop = new CIMObjectPath(args[1]);
```

EXAMPLE 3–18 Deleting a Class *(Continued)*

```
            // delete the class
            cc.deleteClass(cop);
    }
    catch (Exception e) {
            System.out.println("Exception: "+e);
    }
    if (cc != null) {
        cc.close();
        }
     }
}
```

Setting Access Control

You can set access control on a per-user or namespace basis. The following access control classes are stored in the `root\security` namespace:

- `Solaris_Acl` – Base class for Solaris Access Control Lists (ACL). This class defines the string property *capability* and sets its default value to `r` (read only).
- `Solaris_UserAcl` – Represents the access control that a user has to the CIM objects within the specified namespace.
- `Solaris_NamespaceAcl` – Represents the access control on a namespace.

You can set access control for individual users to the CIM objects within a namespace by creating an instance of the `Solaris_UserACL` class and then changing the access rights for that instance. Similarly, you can set access control for a namespace by creating an instance of the `Solaris_NameSpaceACL` class and then using the `createInstance` method to set the access rights for that instance.

An effective way to combine the use of these two classes is to use the `Solaris_NameSpaceACL` class first, to restrict access to all users to the objects in a namespace. Then, you can use the `Solaris_UserACL` class to grant selected users access to the namespace.

The `Solaris_UserAcl` Class

The `Solaris_UserAcl` class extends the `Solaris_Acl` base class, from which it inherits the string property *capability* with a default value of `r` (read only). You can set the *capability* property to any one of the following values for access privileges.

Access Right	Description
r	Read
rw	Read and Write
w	Write
none	No access

The Solaris_UserAcl class defines the following key properties. Only one instance of the namespace and user name ACL pair can exist in a namespace.

Property	Data Type	Purpose
nspace	string	Identifies the namespace to which the ACL applies.
username	string	Identifies the user to which the ACL applies.

▼ To Set Access Control for a User

1. **Create an instance of the** Solaris_UserAcl **class. For example:**

```
...
/* Create a namespace object initialized with root\security
(name of namespace) on the local host. */

CIMNameSpace cns = new CIMNameSpace("", "root\security");

// Connect to the root\security namespace as root.
cc = new CIMClient(cns, user, user_passwd);

// Get the Solaris_UserAcl class
cimclass = cc.getClass(new CIMObjectPath("Solaris_UserAcl");

// Create a new instance of the Solaris_UserAcl
class ci = cimclass.newInstance();
...
```

2. **Set the** *capability* **property to the desired access rights. For example:**

```
...
/* Change the access rights (capability) to read/write for user Guest
on objects in the root\molly namespace.*/
ci.setProperty("capability", new CIMValue(new String("rw"));
ci.setProperty("nspace", new CIMValue(new String("root\molly"));
ci.setProperty("username", new CIMValue(new String("guest"));
...
```

3. **Update the instance. For example:**

```
...
// Pass the updated instance to the CIM Object Manager
cc.createInstance(new CIMObjectPath(), ci);
...
```

The `Solaris_NamespaceAcl` Class

The `Solaris_NamespaceAcl` extends the `Solaris_Acl` base class, from which it inherits the string property *capability* with a default value `r` (read-only for all users). The `Solaris_NamespaceAcl` class defines this key property.

Property	Data Type	Purpose
`nspace`	`string`	Identifies the namespace to which the access control list applies. Only one instance of the namespace ACL can exist in a namespace.

▼ To Set Access Control for a Namespace

1. **Create an instance of the `Solaris_namespaceAcl` class. For example:**

```
...
/* Create a namespace object initialized with root\security
(name of namespace) on the local host. */
CIMNameSpace cns = new CIMNameSpace("", "root\security");

// Connect to the root\security namespace as root.
cc = new CIMClient(cns, user, user_passwd);

// Get the Solaris_namespaceAcl class
cimclass = cc.getClass(new CIMObjectPath("Solaris_namespaceAcl");

// Create a new instance of the Solaris_namespaceAcl
class ci = cimclass.newInstance();
...
```

2. **Set the *capability* property to the desired access rights. For example:**

```
...
/* Change the access rights (capability) to read/write
to the root\molly namespace. */
ci.setProperty("capability", new CIMValue(new String("rw"));
ci.setProperty("nspace", new CIMValue(new String("root\molly"));
...
```

3. **Update the instance. For example:**

```
// Pass the updated instance to the CIM Object Manager
cc.createInstance(new CIMObjectPath(), ci);
```

Working with Qualifiers and Qualifier Types

A CIM qualifier is an element that characterizes a CIM class, instance, property, method, or parameter. Qualifiers have the following attributes:

- Type
- Value
- Name

In MOF syntax, each CIM qualifier must have a CIM qualifier type defined. Qualifiers do not have a scope attribute, which indicates the CIM elements that can use the qualifier. You can only define scope in the qualifier type declaration. You cannot change scope in a qualifier.

The following sample code shows the MOF syntax for a CIM qualifier type declaration. This statement defines a qualifier type named `key`, with a Boolean data type (default value false), which can describe only a property and a reference to an object. The `DisableOverride` flavor means that key qualifiers cannot change their value.

```
Qualifier Key : boolean = false, Scope(property, reference),
                    Flavor(DisableOverride);
```

The following sample code shows the MOF syntax for a CIM qualifier. In this sample MOF file, `key` and description are qualifiers for the property `a`. The property data type is an integer with the property name `a`.

```
{
[key, Description("test")]
int a;
};
```

Getting and Setting CIM Qualifiers

A *qualifier flavor* is a flag that governs the use of a qualifier. Flavors describe rules that specify whether a qualifier can be propagated to derived classes and instances and whether or not a derived class or instance can override the qualifier's original value.

EXAMPLE 3–19 Setting CIM Qualifiers

This example sets a list of CIM qualifiers for a new class to the qualifiers in its superclass.

```
{

 try {
```

EXAMPLE 3–19 Setting CIM Qualifiers *(Continued)*

```
     cimSuperClass = cimClient.getClass(new CIMObjectPath(scName));
        Vector v = new Vector();
        for (Enumeration e = cimSuperClass.getQualifiers().elements();
                        e.hasMoreElements();) {
CIMQualifier qual = (CIMQualifier)((CIMQualifier)e.nextElement()).clone();
        v.addElement(qual);
        }
        cimClass.setQualifiers(v);
 } catch (CIMException exc) {
          return;
        }
    }
}
...
```

Batching Client Requests

You can batch multiple `CIMClient` API calls into a single remote call to reduce the delay introduced by multiple remote message exchanges. You use an instance of the `BatchCIMClient` class to build the list of operations that you want to execute in a batch request. Then use the `performBatchOperations` method of the `CIMClient` class to send the list of operations to the CIM Object Manager.

Note – A batch operation does not imply a single transaction. Each operation is independent of the other operations included in the batch and has no dependencies on the success or failure of the preceding operations.

The `BatchCIMClient` class contains methods that enable you to perform the same CIM operations as in non-batch mode. These methods are similar to `CIMClient` methods except that the `BatchCIMClient` methods do not return the same types as their equivalents in the `CIMClient` class because the values are returned as a list after the batch operation is complete. The methods return an integer operation ID which you can use to get the result of the operation later. As methods of `BatchCIMClient` are invoked, the `BatchCIMClient` object builds a list of `CIMOperation`objects that will be executed later.

When the client executes the batch operation list by invoking CIMClient's `performBatchOperations` method, a `BatchResult` object is returned that contains the results of the batch operation. Clients can then pass the operation ID to the `getResult` method of the `BatchResult` class to get the results of the operations.

If an operation on the list generates an exception, an exception object is embedded in the `BatchResult` object. When you invoke the `getResult` method with the ID of the operation that failed, the exception is thrown by the `getResult` method.

EXAMPLE 3–20 Batching Example

The following example shows how you can use the batching API to perform multiple operations in one remote call. In this example, three operations - enumerateInstanceNames, getClass, and enumerateInstances are performed as a single batch operation.

```
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Vector;
import java.lang.String;

import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;


public class TestBatch {
    public static void main(String args[]) throws CIMException {
           CIMClient cc = null;
           CIMObjectPath cop = null;
           String protocol = CIMClient.CIM_RMI;
           if (args.length < 4) {
               System.out.println("Usage: TestBatch host user passwd
                                   classname " + "[rmi|http]");
               System.exit(1);
               }
           try {
               CIMNameSpace cns = new CIMNameSpace(args[0]);

               UserPrincipal up = new UserPrincipal(args[1]);
               PasswordCredential pc = new PasswordCredential(args[2]);
               if (args.length == 5 && args[4].equalsIgnoreCase("http")) {
                       protocol = CIMClient.CIM_XML;
               }
               cc = new CIMClient(cns, up, pc, protocol);

               CIMObjectPath op = new CIMObjectPath(args[3]);

               BatchCIMClient bc = new BatchCIMClient();
               int[] ids = new int[3];

               ids[0] = bc.enumerateInstanceNames(op);
               ids[1] = bc.getClass(op, false, true, true, null);
               ids[2] = bc.enumerateInstances(op, true, false, false,
                                              false, null);

               BatchResult br = cc.performBatchOperations(bc);
```

EXAMPLE 3–20 Batching Example *(Continued)*

```
            Enumeration instanceNames = (Enumeration)br.getResult
                                        (ids[0]);
                CIMClass cl = (CIMClass)br.getResult(ids[1]);
                Enumeration instances = (Enumeration)br.getResult(ids[2]);

                while (instanceNames.hasMoreElements()) {
                    System.out.println((CIMObjectPath)instanceNames.
                                       nextElement());
                }

                System.out.println(cl.toMOF());

                while (instances.hasMoreElements()) {
                    System.out.println((CIMInstance)instances.
                                       nextElement());
                }

            }
            catch (Exception e) {
                e.printStackTrace();
                System.out.println("Exception: "+e);
            }

            // close session.
            if (cc != null) {
                cc.close();
            }
    }
}
```

Handling CIM Events

Note – For in-depth information on CIM Indications and how they are used to communicate occurrences of events, see the Distributed Management Task Force (DMTF) Event white paper at http://www.dmtf.org/education/whitepapers.php.

An *event* is a real world occurrence. An *indication* is an object that communicates the occurrence of an event. In the Common Information Model, indications are published—not events. Providers generate an indication when an event takes place.

An indication may have zero or more *triggers*, which are recognitions of changes in state. WBEM does not have an explicit object representing a trigger. Instead, a trigger is implied by the following:

- An operation on a basic object of a system — Create, delete, modify, or access a class; modify or access an instance
- Any event that takes place in the managed environment

For example, when a service terminates and a trigger is engaged, this event results in an indication that serves as notification that the service has terminated.

You can view the related CIM event classes in the Solaris WBEM Services schema at `/usr/sadm/lib/wbem/doc/mofhtml/index.html`. The class is structured as follows:

TABLE 3–5 `CIM_Indication` Class Structure

Root Class	Superclass	Subclass
`CIM_Indication`	`CIM_ClassIndication`	`CIM_ClassCreation` , `CIM_ClassDeletion`, `CIM_ClassModification`
	`CIM_InstIndication`	`CIM_InstCreation`, `CIM_InstDeletion`, `CIM_InstMethodCall`, `CIM_InstModification`, `CIM_InstRead`
	`CIM_ProcessIndication`	`CIM_AlertIndication`, `CIM_AlertInstIndication`, `CIM_ThresholdIndication`, `CIM_SNMPTrapIndication`

About Indications

CIM events can be classified as either life cycle or process. A *life cycle event* is a built-in CIM event that occurs in response to a change to data in which a class is created, modified, or deleted, or a class instance is created, modified, deleted, read, or has a method invocation. A *process event* is a user-defined event that is not described by a life cycle event.

Event providers generate indications in response to requests made by the CIM Object Manager. The CIM Object Manager analyzes subscription requests and uses the `EventProvider` and/or the `CIMIndicationProvider` interface to contact the provider, requesting that it generate the appropriate indications. When the provider generates the indication, the CIM Object Manager routes the indication to the destinations specified by the `CIM_IndicationHandler` instances. These instances are created by the subscribers.

Event providers are located in the same manner as instance providers. In the case of subscriptions pertaining to instance life cycle indication (subclasses of `CIM_InstIndication`), once the CIM Object Manager determines the classes covered by the subscription, it contacts the instance providers for those classes. For process indications, the CIM Object Manager contacts the appropriate provider using the `Provider` qualifier.

The CIM Object Manager and the CIM Object Manager Repository handle indications under the following circumstances:

- The CIM Object Manager handles `CIM_InstMethodCall`, `CIM_InstModification`, `CIM_InstDeletion` and `CIM_InstCreation` events if the provider does not support indications, or tells the CIM Object Manager to poll.
- The CIM Object Manager Repository handles all class indications and life cycle indications for classes that do not have providers. These classes include `CIM_ClassCreation`, `CIM_ClassDeletion` `CIM_ClassModification`, `CIM_InstCreation`, `CIM_InstModification`, `CIM_InstDeletion` and `CIM_InstRead`.

In these cases, the provider does not generate indications or implement the `EventProvider` interface. In addition, the provider can delegate event generation responsibilities to the CIM Object Manager. The CIM Object Manager invokes `enumerateInstances` on the providers and compares snapshots of previous states to current states to determine if instances have been created, modified, or deleted.

Note – In most cases, providers should handle their own indications since polling carries a high overhead. In order to generate indications, the provider itself must poll. In this case, the provider can delegate the task to the CIM Object Manager.

If a provider implements the `EventProvider` interface, the CIM Object Manager invokes the methods in the interface and takes actions according to the responses. When the CIM Object Manager determines that a particular provider must participate in a subscription request, the methods are invoked in the following order:

1. `mustPoll` - Invoked by the CIM Object Manager for `CIM_InstCreation`, `CIM_InstDeletion`, and `CIM_InstModification` to determine if the provider wants the CIM Object Manager to poll. If the provider does not implement the `EventProvider` interface, the CIM Object Manager assumes polling by default.
2. `authorizeFilter` - If the provider implements the `Authorizable` interface, this method is invoked by the CIM Object Manager to determine if the subscription is authorized. The provider can make the determination based on the user ID of the owner of the indication handler—the user who receives the indications—or based on the user ID of the user who created the subscription.

   If the provider does not implement the `Authorizable` interface, the CIM Object Manager performs the default read authorization check for the namespace.

If the provider does not implement the `EventProvider` interface and the CIM Object Manager tries to poll, the authorization succeeds if `enumerateInstances` succeeds on the provider.

3. `activateFilter` - Invoked by the CIM Object Manager when the authorization succeeds and the provider does not want to be polled.
4. `deActivateFilter` - Called when a subscription is removed either by the subscriber or the CIM Object Manager; for example, if the destination handler malfunctions.

About Subscriptions

A client application can subscribe to be notified of CIM events. A *subscription* is a declaration of interest in one or more streams of indications. Currently, providers cannot subscribe for event indications.

An application that subscribes for indications of CIM events describes the following:

- The indications to which it wants to subscribe.
- The handler to which the CIM Object Manager delivers the indication.

The occurrence of an event is represented as an instance of one of the subclasses of the `CIM_Indication` class. An indication is generated only when a client subscribes to the event.

▼ To Create a Subscription

An application can create one or more event filters with one or more event handlers. Event indications are not delivered until the application creates the event subscription.

1. **Create an instance of `CIM_Listener`. See Adding a CIM Listener..**
2. **Create an instance of `CIM_IndicationFilter`. See Creating an Event Filter.**
3. **Create an instance of `CIM_IndicationHandler`. See Creating an Event Handler.**
4. **Bind the to the `CIM_IndicationFilter` to the `CIM_IndicationHandler`. See Binding an Event Filter to an Event Handler.**

Adding a CIM Listener

To receive indications of CIM events, first add an instance of `CIMListener` to `CIMClient`, by invoking the `addCIMListener` method on `CIMClient`.

Note – The `CIMListener` interface must implement the `indicationOccured` method, which takes the argument `CIMEvent`. This method is invoked when an indication is available for delivery.

EXAMPLE 3–21 Adding a CIM Listener

```
// Connect to the CIM Object Manager
cc = new CIMClient();

// Register the CIM Listener
cc.addCIMListener(
new CIMListener() {
    public void indicationOccured(CIMEvent e) {
    }
});
```

Creating an Event Filter

Event filters describe the types of events to be delivered and the conditions under which they are delivered. To create an event filter, create an instance of the `CIM_IndicationFilter` class and define values for its properties. Each event filter works only on events that belong to the namespace to which the filter belongs.

The `CIM_IndicationFilter` class has string properties that you can set to uniquely identify the filter, specify a query string, and specify the query language that parses the query string. Currently, only the WBEM Query Language (WQL) is supported.

TABLE 3–6 `CIM_IndicationFilter` Properties

Property	Description	Required/Optional
`SystemCreationClassName`	The name of the system on which the creation class for the filter resides or to which it applies.	Optional. The value is decided by the CIM Object Manager.
`SystemName`	The name of the system on which the filter resides or to which it applies.	Optional. The default for this key property is the name of the system on which the CIM Object Manager is running.
`CreationClassName`	The name of the class or subclass used to create the filter.	Optional. The CIM Object Manager assigns `CIM_IndicationFilter` as the default for this key property.

TABLE 3–6 `CIM_IndicationFilter` Properties *(Continued)*

Property	Description	Required/Optional
`Name`	The unique name of the filter.	Optional. The CIM Object Manager assigns a unique name.
`SourceNamespace`	The path to a local namespace where the CIM indications originate.	Optional. The default is null.
`Query`	A query expression that defines the conditions under which indications are generated. Currently, only Level 1 WBEM Query Language (WQL) expressions are supported. To learn more about WQL query expressions, see Chapter 5.	Required.
`QueryLanguage`	The language in which the query is expressed.	Required. The default is WQL (WBEM Query Language).

▼ To Create an Event Filter

1. **Create an instance of the `CIM_IndicationFilter` class:**

```
CIMClass cimfilter = cc.getClass
        (new CIMObjectPath("CIM_IndicationFilter"),
         true, true, true, null);
CIMInstance ci = cimfilter.newInstance();
```

2. **Specify the name of the event filter:**

```
Name = "filter_all_new_solarisdiskdrives"
```

3. **Create a WQL string to identify that event indications to return:**

```
String filterString = "SELECT *
        FROM CIM_InstCreation WHERE sourceInstance
        ISA Solaris_DiskDrive";
```

4. **Set property values in the `cimfilter` instance to identify the name of the filter, the filter string to select CIM events, and the query language (WQL) to parse the query string.**

```
ci.setProperty("Name", new
        CIMValue("filter_all_new_solarisdiskdrives"));
ci.setProperty("Query", new CIMValue(filterString));
ci.setProperty("QueryLanguage", new CIMValue("WQL");)
```

5. **Create an instance of the `cimfilter` instance called `filter` and store it in the CIM Object Manager Repository:**

```
CIMObjectPath filter = cc.createInstance(new
                                                CIMObjectPath(), ci);
```

EXAMPLE 3–22 Creating an Event Filter

```
CIMClass cimfilter = cc.getClass(new CIMObjectPath
                                 ("CIM_IndicationFilter"), true);
CIMInstance ci = cimfilter.newInstance();
//Assuming that the test_a class exists in the namespace
String filterString = "select * from CIM_InstCreation where sourceInstance
                       isa test_a"

ci.setProperty("query", new CIMValue(filterString));
CIMObjectPath filter = cc.createInstance(newCIMObjectPath(), ci);
```

Creating an Event Handler

An event handler is an instance of a `CIM_IndicationHandler` class. You set the properties in an instance of the `CIM_IndicationHandler` class to uniquely name the handler and identify the UID of its owner. The CIM Event MOF defines a `CIM_IndicationHandlerCIMXML` class for describing the destination for indications to be delivered to client applications using the HTTP protocol. The Solaris Event MOF extends the `CIM_IndicationHandler` class by creating the `Solaris_JAVAXRMIDelivery` class to handle delivery of indications of CIM events to client applications using the RMI protocol. RMI clients must instantiate the `Solaris_JAVAXRMIDelivery` class to set up an RMI delivery location.

TABLE 3–7 `CIM_IndicationHandler` Properties

Property	Description	Required/Optional
SystemCreationClassName	The name of the system on which the creation class for the handler resides or to which it applies.	Optional. Completed by the CIM Object Manager.
SystemName	The name of the system on which the handler resides or to which it applies.	Optional. The default value for this key property is the name of the system on which the CIM Object Manager is running.
CreationClassName	The class or subclass used to create the handler.	Optional. The CIM Object Manager assigns the appropriate class name as the default for this key property.

TABLE 3–7 `CIM_IndicationHandler` Properties *(Continued)*

Property	Description	Required/Optional
Name	The unique name of the handler.	Optional. The client application must assign a unique name.
Owner	The name of the entity that created or maintains this handler. The provider can check this value to determine whether or not to authorize a handler to receive an indication.	Optional. The default value is the Solaris user name of the user creating the instance.

EXAMPLE 3–23 Creating an Event Handler

```
// Create an instance of the Solaris_JAVAXRMIDelivery class or get
// the appropriate instance of the handler.
CIMInstance ci = cc.getIndicationHandler(null);

//Create a new instance (delivery) from
//the rmidelivery instance.
CIMObjectPath delivery = cc.createInstance(new CIMObjectPath(), ci);
```

Binding an Event Filter to an Event Handler

You bind an event filter to an event handler by creating an instance of the `CIM_IndicationSubscription` class. When you create an indication of this class, indications for the events specified by the event filter are delivered.

The following example creates a subscription (`filterdelivery`) and defines the `filter` property to the `filter` object path created in "To Create an Event Filter" on page 79, and defines the `handler` property to the `delivery` object path created in Example 3–23.

EXAMPLE 3–24 Binding an Event Filter to an Event Handler

```
CIMClass filterdelivery = cc.getClass(new
        CIMObjectPath("CIM_IndicationSubscription"),
        true, true, true, null);
ci = filterdelivery.newInstance():

//Create a property called filter that refers to the filter instance.
ci.setProperty("filter", new CIMValue(filter));

//Create a property called handler that refers to the delivery instance.
ci.setProperty("handler", new CIMValue(delivery));

CIMObjectPath indsub = cc.createInstance(new CIMObjectPath(), ci);
```

EXAMPLE 3–24 Binding an Event Filter to an Event Handler *(Continued)*

Reading and Writing Log Messages

The Solaris MOFs include logging classes. Clients can create and read log records using these classes to record errors, warnings, and informational messages. For example, a log message can indicate when a system cannot access a serial port, when a system successfully mounts a file system, or when the number of processes that are running on a system exceeds the allowed number.

The underlying providers for the logging classes can forward logging requests to the syslog daemon, the default logging system in the Solaris operating environment. See `syslogd`(1M).

About Log Files

WBEM log messages are stored in individual log files in the `/var/sadm/wbem/log` directory. The names of the log files, the directory in which the log files are stored, the log file size limit, the number of log files to store, and whether to forward messages to `syslogd`(1M) are properties that you manipulate with the `singleton` instance of the `Solaris_LogServiceProperties` class.

The format of each log entry is defined by the `Solaris_LogEntry` class, which is a subclass of `CIM_LogRecord`. You can find `Solaris_LogEntry` in `Solaris_Device1.0.mof`, and `CIM_LogRecord` in `CIM_Device26.mof`.

A log message includes the following elements:

TABLE 3–8 Log Message Elements

Element	Description
`Category`	Type of message – application, system, or security
`Severity`	Severity of the condition – warning or error
`Application`	Name of the application (or the provider) that is writing the log message
`User`	Name of the user who was using the application when the log message was generated

TABLE 3–8 Log Message Elements *(Continued)*

Element	Description
`Client Machine`	Name and IP address of the system that the user was on when the log message was generated
`Server Machine`	Name of the system on which the incident that generated the log message occurred
`Summary Message`	Descriptive summary of the incident
`Detailed Message`	Detailed description of the incident
`Data`	Contextual information that provides a better understanding of the incident
`SyslogFlag`	Boolean flag that specifies whether or not to send the message to `syslogd`(1M)

EXAMPLE 3–25 Creating an Instance of `Solaris_LogEntry`

This example creates an instance of `Solaris_LogEntry`and sets the instance.

```
public class CreateLog {
    public static void main(String args[]) throws CIMException {

        // Display usage statement if insufficient command line
        // arguments are passed.
        if (args.length < 3) {
            System.out.println("Usage: CreateLog host username password
                               " + "[rmi|http]");
            System.exit(1);
        }

        String protocol = CIMClient.CIM_RMI;
        CIMClient cc = null;
        CIMObjectPath cop = null;
        BufferedReader d = new BufferedReader(new InputStreamReader
                                              (System.in));

        String input_line = "";

        // Query user for number of records that need to be created.
        System.out.print("How many log records do you want to write? ");
        int num_recs = 0;

        try {
                num_recs = Integer.parseInt(d.readLine());
        } catch (Exception ex) {
                ex.printStackTrace();
                System.exit(1);
        }

        // Over-arching try-catch block
```

EXAMPLE 3–25 Creating an Instance of `Solaris_LogEntry` *(Continued)*

```
try {
    CIMNameSpace cns = new CIMNameSpace(args[0]);
    UserPrincipal up = new UserPrincipal(args[1]);
    PasswordCredential pc = new PasswordCredential(args[2]);

    // Set up the transport protocol - set by default to RMI.
    if (args.length == 4 && args[3].equalsIgnoreCase("http")) {
        protocol = CIMClient.CIM_XML;
    }

    cc = new CIMClient(cns, up, pc, protocol);

        Vector keys = new Vector();
        CIMProperty logsvcKey = null;


        // Prompt user for relevant info needed to create the
        // log record.

        System.out.println("Please enter the record Category: ");
        System.out.println("\t(0)application, (1)security,
                                              (2)system");
        logsvcKey = new CIMProperty("category");
        input_line = d.readLine();
        logsvcKey.setValue(new CIMValue(Integer.valueOf
                                    (input_line)));
        keys.addElement(logsvcKey);
        System.out.println("Please enter the record Severity:");
        System.out.println("\t(0)Informational, (1)Warning,
                                               (2)Error");
        logsvcKey = new CIMProperty("severity");
        input_line = d.readLine();
        logsvcKey.setValue(new CIMValue(Integer.valueOf
                          (input_line)));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("Source");
        System.out.println("Please enter Application Name:");
        logsvcKey.setValue(new CIMValue(d.readLine()));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("SummaryMessage");
        System.out.println("Please enter a summary message:");
        logsvcKey.setValue(new CIMValue(d.readLine()));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("DetailedMessage");
        System.out.println("Please enter a detailed message:");
        logsvcKey.setValue(new CIMValue(d.readLine()));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("RecordData");
        logsvcKey.setValue(
                new CIMValue("0xfe 0x45 0xae 0xda random data"));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("SyslogFlag");
        logsvcKey.setValue(new CIMValue(new Boolean(true)));
```

EXAMPLE 3–25 Creating an Instance of `Solaris_LogEntry` *(Continued)*

```
                keys.addElement(logsvcKey);
                CIMObjectPath logreccop =
                        new CIMObjectPath("Solaris_LogEntry", keys);
                CIMClass logClass = cc.getClass(logreccop);
                CIMInstance ci = logClass.newInstance();
                ci.setClassName("Solaris_LogEntry");
                ci.setProperties(keys);
                // System.out.println(ci.toString());

                // Create as many instances of the record as requested.
                for (int i = 0; i < num_recs; i++) {
                        cc.createInstance(logreccop, ci);
                }
        } catch (Exception e) {
            System.out.println("Exception: "+e);
                e.printStackTrace();
        }

        // close session.
        if (cc != null) {
            cc.close();
        }
    }
}
```

EXAMPLE 3–26 Displaying a List of Log Records

This example displays a list of log records.

```
public class ReadLog {
   public static void main(String args[]) throws CIMException {

        String protocol = CIMClient.CIM_RMI;

        // Display usage statement if insufficient command line
        // arguments are passed.
        if (args.length < 3) {
            System.out.println("Usage: ReadLog host username password " +
                               "[rmi|http]");
            System.exit(1);
        }

        CIMClient cc = null;
        CIMObjectPath cop = null;
        CIMObjectPath serviceObjPath = null;
        Vector inVec = new Vector();
        Vector outVec = new Vector();

        // Over-arching try-catch block
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal(args[1]);
```

EXAMPLE 3–26 Displaying a List of Log Records *(Continued)*

```
            PasswordCredential pc = new PasswordCredential(args[2]);

            // Set up the transport protocol - set by default to RMI.
            if (args.length == 4 && args[3].equalsIgnoreCase("http")) {
                protocol = CIMClient.CIM_XML;
            }

            cc = new CIMClient(cns, up, pc, protocol);

            cop = new CIMObjectPath("Solaris_LogEntry");

            // Enumerate the list of instances of class Solaris_LogEntry
            Enumeration e = cc.enumerateInstances(cop, true, false,
                                                  false, false, null);

            // iterate over the list and print out each property.
            for (; e.hasMoreElements(); ) {
                System.out.println("--------------------------------");
                CIMInstance ci = (CIMInstance)e.nextElement();
                System.out.println("Log filename : " +
                    ((String)ci.getProperty("LogName").getValue().
                                                        getValue()));
                int categ =
    (((Integer)ci.getProperty("Category").getValue().getValue()).
       intValue());
                if (categ == 0)
                    System.out.println("Category : Application Log");
                else if (categ == 1)
                    System.out.println("Category : Security Log");
                else if (categ == 2)
                    System.out.println("Category : System Log");
                int severity =
    (((Integer)ci.getProperty("Severity").getValue().getValue()).
       intValue());
                if (severity == 0)
                    System.out.println("Severity : Informational");
                else if (severity == 1)
                    System.out.println("Severity : Warning Log!");
                else if (severity == 2)
                    System.out.println("Severity : Error!!");
                System.out.println("Log Record written by :" +
    ((String)ci.getProperty("Source").getValue().getValue()));
                System.out.println("User : " +
    ((String)ci.getProperty("UserName").getValue().getValue()));
                System.out.println("Client Machine : " +
    ((String)ci.getProperty("ClientMachineName").getValue().getValue()));
                System.out.println("Server Machine : " +
    ((String)ci.getProperty("ServerMachineName").getValue().getValue()));
                System.out.println("Summary Message : " +
    ((String)ci.getProperty("SummaryMessage").getValue().getValue()));
                System.out.println("Detailed Message : " +
    ((String)ci.getProperty("DetailedMessage").getValue().getValue()));
                System.out.println("Additional data : " +
```

EXAMPLE 3–26 Displaying a List of Log Records *(Continued)*

```
    ((String)ci.getProperty("RecordData").getValue().getValue()));
                boolean syslogflag =
((Boolean)ci.getProperty("SyslogFlag").getValue().getValue()).
  booleanValue();
                if (syslogflag == true) {
                    System.out.println("Record was written to syslog");
                } else {
                    System.out.println("Record was not written to syslog");
                }
                System.out.println("--------------------------------");
            }
        } catch (Exception e) {
            System.out.println("Exception: "+e);
            e.printStackTrace();
        }

        // close session.
        if (cc != null) {
            cc.close();
        }
    }
}
```

CHAPTER 4

Writing a Provider Program

This chapter describes how to write a provider program, and includes the following topics:

- "About Providers" on page 89
- "Implementing the Provider Interfaces" on page 93
- "Creating a Provider" on page 103

Note – For detailed information on the WBEM provider APIs (`javax.wbem.provider`), see `file:/usr/sadm/lib/wbem/doc/index.html`.

About Providers

Providers are special classes that communicate with managed resources, such as disk drives and CPUs, to access data and then forward the data to the CIM Object Manager for integration and interpretation. They can relieve the CIM Object Manager, the primary WBEM agent that coordinates Solaris WBEM Services, by assuming the task of managing distinct subsets of WBEM resources. Providers use the `javax.wbem.provider` API to transfer this data. When the CIM Object Manager receives a request for data from an application that is not available in the CIM Object Manager Repository, it forwards the request, using the provider interfaces, to the appropriate provider.

Solaris software providers exist for a variety of areas: users, groups, aliases, roles, file systems, disks, processes, cron tool, network configuration, product registry, and device/system performance monitoring.

Providers create, modify, and delete *instances* rather than classes (which serve as templates for the instances). Instances can exist in persistent storage or be used dynamically.

Although providers have their own process and memory and perform work delegated by the CIM Object Manager, the CIM Object Manager must know the location of each provider in order to perform its task of coordinating WBEM. You inform the CIM Object Manager about new or modified providers by including those providers in a MOF file. A MOF file defines the classes and instances that a provider supports. You register a MOF file using the `mofcomp`(1M) command.

Providers do the following:

- **Provide data to management applications** – When a management application requests data about a managed resource that is not available in the CIM Object Manager Repository, the CIM Object Manager forwards the request to a provider. The provider accesses the data from the managed resource and passes the data back to the CIM Object Manager. If the data received from a managed resource is in a native format (such as C code), the provider maps the data to Java CIM classes prior to passing it to the CIM Object Manager.
- **Control management resources** – When a management application sends data to the CIM Object Manager to control a managed resource, the CIM Object Manager passes the data to the appropriate provider. If the managed resource requires data in a native format, the provider maps the CIM classes to the resource's native format prior to passing it along.

Note – Providers must reside on the same machine as the CIM Object Manager.

Provider Data Sources

Providers can retrieve data from the following:

- **Non-persistent data** – Variables that are local to the provider class that exist only when the provider's methods are run.
- **Persistent memory that is local to the provider** – Used by creating global variables in the provider class. This provider memory is erased when the CIM Object Manager is stopped and restarted.
- **The CIM Object Manager Repository** – This persistent memory is erased when Solaris WBEM Services is uninstalled. The provider must use CIM Object Manager handles and an internal provider to access this memory through the CIM Object Manager.
- **Files and databases maintained by the provider, or dynamic data** – Providers can generate data dynamically by retrieving it from a system. For example, a provider can make a system call to retrieve the number of processes currently running.

Types of Providers

Providers are categorized according to the types of requests they service. Client programs communicate with the CIM Object Manager (and access WBEM data) through the client API (see `file:/usr/sadm/lib/wbem/doc/index.html`). The CIM Object Manager maps the provider methods to the corresponding client methods in the client API. However, the argument lists and return values of corresponding methods may differ:

- If a provider stores data in the CIM Object Manager Repository, then it accesses the Repository using handles to the CIM Object Manager (see "Implementing the Provider Interfaces" on page 93), which call the methods of the client API.
- If a provider needs to create instances or associations in the CIM Object Manager Repository, then it uses an internal provider (see "Implementing the Provider Interfaces" on page 93), which calls methods of Instance or Associator Providers that are internal to WBEM.

Make sure your argument list and return type is correct for the method and class used.

The Solaris WBEM SDK provider types are shown in the following table.

TABLE 4–1 Provider Types

Type	Class Name	Description
Instance	`CIMInstanceProvider`	Supply dynamic instances of a given class, and support instance retrieval, enumeration, modification, and deletion.
Method	`MethodProvider`	Supply methods of one or more classes.
Associator	`CIMAssociatorProvider`	Supply instances of dynamic association classes.
Indication	`EventProvider`	Handle indications of CIM events.
Authorizable	None	A marker interface that indicates to the CIM Object Manager that the provider does its own authorization check.

A single provider can act as one or more of the above types by registering and implementing the relevant methods.

Provider Naming Conventions

You can include providers in a single Java class file or store each provider in a separate class. The provider name identifies the Java class that serves as the provider for the class. Currently, the CIM Object Manager supports only providers written in the Java language.

Provider and class names must follow these rules:

- The class name must be a valid CIM class, that is, that the name contains a prefix of characters, followed by an underscore, followed by more characters.

  For example, `green_apples` and `red_apples` are valid CIM class names, whereas `apples`, `apples_`, and `_apples` are not.
- The provider name specified in the MOF file must match the name of the provider class file.

  For example, `SimpleCIMInstanceProvider` is the provider and `Ex_SimpleCIMInstanceProvider` is the class.

Note – You must prepend "`java:`" to every provider qualifier to notify the CIM Object Manager that the provider is written in the Java language.

Follow standard Java class and package naming conventions to create your provider names. The prefix of a package name is written in lowercase ASCII letters and must be one of the top-level domain names (`com`, `edu`, `gov`, `mil`, `net`, `org`), or one of the English two-letter country codes specified in ISO Standards 3166, 1981.

Subsequent components of the package name will vary according to your organization's internal naming conventions. Such conventions might specify that certain directory name components are division, department, project, machine, or login names. For example, the provider name `java:com.sun.wbem.cimom` indicates the following:

- `java:` – Language used to write the provider
- `com` – Top-level domain name
- `sun` – Company name
- `wbem` – Product name
- `cimom` – Type of class files that implement the CIM Object Manager

Implementing the Provider Interfaces

When you write a provider, you must determine the interfaces that your provider will support. You must implement all the methods of each interface that your provider supports. In addition, every provider must implement the `CIMProvider` interface, which has two methods:

- `initialize(CIMOMHandle` *cimom*`)` – If your provider stores data in the CIM Object Manager Repository, it must assign the passed CIM Object Manager handle to the CIM Object Manager handle that it will use to contact the CIM Object Manager. For example:

```
private CIMOMHandle cimom = null;
      ...
    public void initialize(CIMOMHandle cimom) throws CIMException {
      this.cimom = (CIMOMHandle) cimom;
```

  The provider creates instances or manipulates associations in the CIM Object Manager Repository, it must first cast the passed CIM Object Manager handle to the subclass `ProviderCIMOMHandle`, and then fetch an internal instance or association provider. For example:

```
private ProviderCIMOMHandle cimom = null;
      private CIMAssociatorProvider ap = null;
      ...
    public void initialize(CIMOMHandle cimom) throws CIMException {
      this.cimom = (ProviderCIMOMHandle) cimom;
      ap = pcimom.getInternalProvider();
```

  Note – The `initialize` command automatically runs each time a provider is initialized after the CIM Object Manager restarts.

- `cleanup()` – Currently acts as a placeholder.

Writing an Instance Provider

This following sample code implements the `enumerateInstances` and `getInstance` interfaces for the `Ex_SimpleCIMInstanceProvider` class. For brevity, this example implements the `deleteInstance`, `createInstance`, `setInstance`, and `execQuery` interfaces by throwing a `CIMException`.

Note – For information on implementing the `execQuery` method, see "Parsing Queries" on page 111.

EXAMPLE 4–1 CIMInstance Provider

```
/*
 * "@(#)SimpleCIMInstanceProvider.java"
 */
import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.provider.CIMProvider;
import javax.wbem.provider.CIMInstanceProvider;
import javax.wbem.provider.MethodProvider;
import java.util.*;
import java.io.*;

public class SimpleCIMInstanceProvider implements CIMInstanceProvider{
     static int loop = 0;
     public void initialize(CIMOMHandle cimom) throws CIMException {
     }
     public void cleanup() throws CIMException {
     }
     public CIMObjectPath[] enumerateInstanceNames(CIMObjectPath op,
                                                   CIMClass cc)
     throws CIMException {
               return null;
     }
     /*
      * enumerateInstances:
      * The entire instances and not just the names are returned.
      */
     public CIMInstance[] enumerateInstances(CIMObjectPath op,
                          boolean localOnly,boolean includeQualifiers,
                          boolean includeClassOrigin,String[]
                          propertyList, CIMClass cc) throws CIMException
     {
     if (op.getObjectName().equalsIgnoreCase\
                            ("Ex_SimpleCIMInstanceProvider"))
             {
             Vector instances = new Vector();
             CIMInstance ci = cc.newInstance();
                 if (loop == 0){
                     ci.setProperty("First", new CIMValue("red"));
                     ci.setProperty("Last", new CIMValue("apple"));
                     // only include the properties that were requested
       ci = ci.filterProperties(propertyList, includeQualifier,
       includeClassOrigin);
                     instances.addElement(ci);
                     loop += 1;
                 } else {
                     ci.setProperty("First", new CIMValue("red"));
                     ci.setProperty("Last", new CIMValue("apple"));
                     // only include the properties that were requested
         ci = ci.filterProperties(propertyList, includeQualifier,
         includeClassOrigin);
                     instances.addElement(ci);
                     ci = cc.newInstance();
                     ci.setProperty("First", new CIMValue("green"));
```

EXAMPLE 4–1 `CIMInstance` Provider *(Continued)*

```
                ci.setProperty("Last", new CIMValue("apple"));
                // only include the properties that were requested
   ci = ci.filterProperties(propertyList, includeQualifier,
   includeClassOrigin);
                instances.addElement(ci);
            }
          return (CIMInstance[])instances.toArray();
          }
      throw new CIMException(CIM_ERR_INVALID_CLASS);
 }

 public CIMInstance getInstance(CIMObjectPath op, boolean localOnly,
 boolean includeQualifiers, boolean includeClassOrigin,
 String[] propertyList, CIMClass cc) ) throws CIMException {
         if (op.getObjectName().equalsIgnoreCase
                         ("Ex_SimpleCIMInstanceProvider"))
         {
             CIMInstance ci = cc.newInstance();
// we need to get the keys from the passed in object path, this
// will uniqeuly identify the instance we want to get
java.util.Vector keys = cop.getKeys();
// Since this is a contrived example we will simply place the keys
// into the instance and be done.
             ci.setProperties(keys);
// if we had other non-key properties we should add them here.

             // only include the properties that were requested
ci = ci.filterProperties(propertyList, includeQualifiers,
includeClassOrigin);
             return ci;
         }
        throw new CIMException(CIM_ERR_INVALID_CLASS);
  }

 public CIMInstance[] execQuery(CIMObjectPath op, \
                       String query, String ql, CIMClass cc)
       throws CIMException {
       throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
}

 public void setInstance(CIMObjectPath op, CIMInstance ci, boolean
                        includeQualifiers, String[] propertyList)
       throws CIMException {
       throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
}

 public CIMObjectPath createInstance(CIMObjectPath op, CIMInstance ci)
          throws CIMException {
       throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
}

 public void deleteInstance(CIMObjectPath cp) throws CIMException {
     throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
```

EXAMPLE 4–1 `CIMInstance` Provider *(Continued)*

```
    }
}
```

Writing a Method Provider

The method `invokeMethod` is the *only* way that a client program can call the methods of Solaris WBEM providers, whether those providers are:

- **Built–in** – The "platform-free" `CIM_*` providers or the Solaris-specific `Solaris_*` providers.
- **Added by developers** – For example, a method provider, whether it supplies provider or non-WBEM methods, is created by implementing the `MethodProvider` interface.

The following sample code creates the `Solaris_ComputerSystem` provider class that routes requests from the CIM Object Manager to one or more specialized providers. These providers service requests for dynamic data for a particular type of managed object. For example, the `Solaris_Package` provider services requests to execute methods in the `Solaris_Package` class.

The method provider implements a single method, `invokeMethod`, that calls the appropriate provider to either reboot a system, shut down a system, or delete a serial port.

EXAMPLE 4–2 Method Provider

```
...
public class Solaris_ComputerSystem implements MethodProvider {
    ProviderCIMOMHandle pch = null;
    public void initialize(CIMOMHandle ch) throws CIMExcepiton {
        pch = (ProviderCIMOMHandle)ch;
    }

    public void cleanup() throws CIMException {
    }

    public CIMValue invokeMethod(CIMObjectPath op, String methodName,
            Vector inParams, Vector outParams)  throws CIMException {
        if (op.getObjectName().equalsIgnoreCase("solaris_computersystem")) {
            if (methodName.equalsIgnoreCase("reboot")) {
                // call helper function, not shown here
                return new CIMValue(rebootSystem());
            }
            if (methodName.equalsIgnoreCase("shutdown")) {
                // call helper function, not shown here
                return new CIMValue(shutdownSystem());
            }
        }
```

EXAMPLE 4–2 Method Provider *(Continued)*

```
        if (op.getObjectName().equalsIgnoreCase("solaris_serialport")) {
            if (methodName.equalsIgnoreCase("disableportservice")) {
                // call helper function, not shown here
                return new CIMValue(deletePort(op));
            }
        }
        // error if we get here
        throw new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED,
            "The requested function does not exist");
    }
    // helper functions would be defined below
    ...
}
```

Writing an Associator Provider

Note – The `objectName` argument in each of the association methods called by your client program, that is, `CIMObjectPath`, must be the object path of an *instance*, not a class.

Unless the CIM Object Manager sees the object path of an instance, it assumes that the client wants to see the class definitions of the association (the templates from which the association's member instances are derived) in the CIM Object Manager Repository, and will use the client API's association method and not that of the provider's.

The most important part of designing and coding an association is the association class itself. Your association will only be as complex as the contents of the association class. The number of members of the association equals the number of references in the association class. Roles can be used to model more complicated associations. Following are some sample association classes:

- An asymmetrical pair relationship, such as a one-to-one relationship between a teacher and a student, with two roles defined (`teaches` and `taughtby`):

```
class TeacherStudent
    {
        Teacher REF teaches;
        Student REF taughtby;
    };
```

- A one-to-many relationship:

```
class Classroom
    {
        Teacher REF teaches;
```

```
        Student1 REF taughtby;
        Student2 REF taughtby;
        Student3 REF taughtby;
        Student4 REF taughtby;
    };
```

- A many-to-many relationship:

```
class TeachingAssistants
    {
        Assistant1 REF assists;
        Assistant2 REF assists;
        Student1 REF assistedby;
        Student2 REF assistedby;
        Student3 REF assistedby;
        Student4 REF assistedby;
        Student5 REF assistedby;
    };
```

- An association of more than two members of equal standing:

```
class Club
    {
        Member1 REF;
        Member2 REF;
        Member3 REF;
    };
```

The following code sample implements the `associators` method. The CIM Object Manager passes values for `associatorNames`, `objectName`, `role`, `resultRole`, `includeQualifiers`, `includeClassOrigin`, and `propertyList` to the association provider. In addition, the code prints the name of the CIM associator class and the CIM class or instance whose associated objects are to be returned. This provider handles instances of `example_teacher` and `example_student` classes.

EXAMPLE 4–3 `CIMAssociator` Provider

```
...

public CIMInstance[] associators(CCIMObjectPath assocName, CIMObjectPath
                objectName, String resultClass, String role, String
                resultRole, boolean includeQualifiers, boolean
                includeClassOrigin, String[] propertyList)
                throws CIMException {
        System.out.println("Associators "+assocName+" "+objectName);
      if (objectName.getObjectName()equalsIgnoreCase("example_teacher")) {
      Vector v = new Vector();
      if ((role != null)  && (!role.equalsIgnoreCase("teaches"))) {
        // Teachers only play the teaches role.
            return v;
        }
        if ((resultRole != null)  && (!resultRole.equalsIgnoreCase
                                    ("taughtby"))) {
        // Teachers only result in taughtby role
        return v;
        }
```

EXAMPLE 4–3 `CIMAssociator` Provider *(Continued)*

```
        // Get the associators of a teacher
        CIMProperty nameProp = (CIMProperty)objectName.getKeys().elementAt
                               (0);
        String name = (String)nameProp.getValue().getValue();
        // Get the student class
        CIMObjectPath tempOp = new CIMObjectPath("example_student");
        tempOp.setNameSpace(assocName.getNameSpace());
        CIMClass cc = cimom.getClass(tempOp, false);
        // Test the instance name passed by objectName
        // and return the associated instances of the student class.
        if(name.equals("teacher1")) {
                // Get students for teacher1
                CIMInstance ci = cc.newInstance();
                ci.setProperty("name", new CIMValue("student1"));
                v.addElement(ci.filterProperties(propertyList,
                             includeQualifiers,
                        includeClassOrigin));
                ci = cc.newInstance();
                ci.setProperty("name", new CIMValue("student2"));
                v.addElement(ci.filterProperties(propertyList,
                        includeQualifiers, includeClassOrigin));
                return v;
        }
}
```

Writing an Indication Provider

To generate an indication for a CIM event, do the following:

- Use the methods in the `EventProvider` interface to detect when to start and stop delivering indications of the CIM event.
- Create an instance of one or more subclasses of the `CIM_Indication` class to store information about the CIM event that occurred.
- Use the `deliverEvent` method in the `ProviderCIMOMHandle` interface to deliver indications to the CIM Object Manager.

▼ How To Generate an Event Indication

1. **Implement the `EventProvider` interface.**

   For example:

```
public class sampleEventProvider
implements InstanceProvider EventProvider{

    // Reference for provider to contact the CIM Object Manager
    private ProviderCIMOMHandle cimom;
   }
```

2. **Execute each of the methods listed in Table 4–2 for each instance indication that the provider handles.**

3. **Create an indication for each create, modify, and delete instance event type.**

   For example, in the `createInstance` method:

```
public CIMObjectPath createInstance(CIMObjectPath op,
        CIMInstance ci)
    throws CIMException {
        CIMObjectpath newop = ip.createInstance(op, ci);
        CIMInstance indication = new CIMInstance();
        indication.setClassName("CIM_InstCreation");
        CIMProperty cp = new CIMProperty();
        cp.setName("SourceInstance");
        cp.setValue(new CIMValue(ci));
        Vector v = new Vector();
        v.addElement(cp);
        indication.setProperties(v);
        ...
    }
```

4. **Deliver the event indication to the CIM Object Manager:**

```
cimom.deliverEvent(op.getNameSpace(), indication);
```

Event Provider Methods

An event provider implements the `EventProvider` interface. This interface contains methods that the CIM Object Manager uses to notify the provider when a client has subscribed for indications of CIM events, and when a client has cancelled the subscription for CIM events. These methods also allow the provider to indicate whether or not the CIM Object Manager should poll for some event indications and whether or not the provider should authorize the return of an indication to a handler.

The following table lists the methods in the `EventProvider` interface that must be implemented by an event provider.

TABLE 4–2 `EventProvider` Methods

Method	Description
`activateFilter`	When a client creates a subscription, the CIM Object Manager calls this method to ask the provider to check for CIM events.
`authorizeFilter`	When a client creates a subscription, the CIM Object Manager calls this method to test if the specified filter expression is allowed.

TABLE 4–2 `EventProvider` Methods *(Continued)*

Method	Description
`deActivateFilter`	When a client removes a subscription, the CIM Object Manager calls this method to ask the provider to deactivate the specified event filter.
`mustPoll`	When a client creates a subscription, the CIM Object Manager calls this method to test if the specified filter expression is allowed by the provider, and if it must be polled.

The CIM Object Manager passes values for the following arguments to all methods:

- *filter* – `SelectExp` that specifies the CIM events for which indications must be generated.
- *eventType* – `String` that specifies the type of CIM event, which can also be extracted from the FROM clause of the select expression.
- *classPath* – `CIMObjectPath` that specifies the name of the class for which the event is required.

In addition, the `activateFilter` method takes the boolean `firstActivation`, indicating that this is the first filter for this event type. The `deActivateFilter` method takes the boolean `lastActivation`, indicating that this is the last filter for this event type.

Creating and Delivering Indications

When a client application subscribes for indications of CIM events by creating an instance of the `CIM_IndicationSubscription` class, the CIM Object Manager forwards the request to the appropriate provider. If the provider implements the `EventProvider` interface, the CIM Object Manager notifies the provider when to start sending indications for the specified events by calling the provider's `activateFilter` method. In addition, the CIM Object Manager notifies the provider when to stop sending indications for the specified events by calling the provider's `deActivateFilter` method.

The provider responds to the CIM Object Manager's requests by creating and delivering an indication each time the provider creates, modifies, or deletes an instance. A provider typically defines a flag variable that is set when the CIM Object Manager calls the `activateFilter` method that is cleared when the CIM Object Manager calls the `deActivateFilter` method. Then in each method that creates, modifies, or deletes an instance, the provider checks the status of the activate filter flag. If the flag is set, the provider creates an indication containing the created CIM instance object and uses the `deliverEvent` method to return the indication to the CIM Object Manager. If the flag is not set, the provider does not create and deliver an indication of the event.

A provider starts delivering indications when the `activateFilter` method is called. The provider creates instances of concrete subclasses of `CIM_Indication` and invokes the `ProviderCIMOMHandled.deliverIndication` method. The CIM Object Manager receives the indication and delivers the indication to the appropriate indication handlers. A provider can handle multiple event types. For example, in the case of life cycle indications, a provider can handle `CIM_InstCreation`, `CIM_InstDeletion`, and `CIM_InstModification`.

To keep track of types that have subscriber interest, the provider can use the `firstActivation` and `lastActivation` flags passed in the `activateFilter` and `deActivateFilter` calls, respectively. The `firstActivation` flag is true when the subscription is the first one for the particular event type. Similarly, `lastActivation` is true when the last subscription for the particular event type is removed. By checking these flags, the provider can easily allocate or deallocate resources to monitor the specified event types.

About Authorizations

A provider that handles sensitive data can check authorizations for requests for indications. The provider must implement the `Authorizable` interface to indicate that it handles authorization checking. The provider also implements the `authorizeFilter` method. The CIM Object Manager calls this method to test if the owner (UID) of an event handler is authorized to receive the indications that result from evaluating a filter expression. The UID for the owner of the event destination (event handler) can be different than the owner of the client application requesting the filter activation.

Writing a Native Provider

Providers get information from and set information on managed devices. A *native provider* is a program specifically written for a particular managed device. For example, a provider that accesses data on a Solaris system usually includes C functions to query the system.

The common reasons for writing a native provider are as follows:

- **Efficiency** – You may want to implement a small portion of time-critical code in a lower-level programming language, such as Assembly, and then have your Java application call these functions.
- **Need to access platform-specific features** – The standard Java class library might not support the platform-dependent features required by your application.
- **Legacy code** – You want to continue to use your legacy code with a Java provider.

The Java Native Interface is part of the JDK software. By writing programs using the Java Native Interface, you ensure that your code is completey portable across all platforms.The Java Native Interface enables Java code that runs within a Java virtual machine to operate with applications and libraries written in other languages, such as C, C++, and assembly.

For more information on writing and integrating Java programs with native methods, visit the Java Web site at `http://java.sun.com`.

Creating a Provider

Follow these steps to create a provider.

1. Create or edit your provider program.
2. Compile the Java program to create the class files.
3. Copy any shared object files (`.so`) to `/usr/sadm/lib/wbem`.
4. Set your CLASSPATH to the location of your `.class` and `.jar` files.
5. Register the provider.

▼ How to Set the Provider `CLASSPATH`

You set the provider `CLASSPATH` to tell the CIM Object Manager where the `.class` and `.jar` files are located.

1. **Create an instance of the `Solaris_ProviderPath` class.**

   For example:

```
/* Create a namespace object initialized with root\system
(name of namespace) on the local host. */
CIMNameSpace cns = new CIMNameSpace("", "root\system");

// Connect to the root\system namespace as root.
cc = new CIMClient(cns, "root", "root_password");

// Get the Solaris_ProviderPath class
cimclass = cc.getClass(new CIMObjectPath("Solaris_ProviderPath");

// Create a new instance of Solaris_ProviderPath.
class ci = cimclass.newInstance();
```

2. **Set the *pathurl* property to the location of the files using standard URL format.**

   For example:

```
/* Set the provider CLASSPATH to /myhome/myproviders */
ci.setProperty("pathurl", new CIMValue(new String
                          ("file:///myhome/myproviders/")));
```

Standard URL format is shown in the following table:

Provider `CLASSPATH`	**Standard URL Format**
Absolute path to directory	`file:///a/b/c/`
Absolute path to `.jar` file	`file:///a/b/my.jar`

3. **Create the instance.**

   For example:

```
// Pass the updated instance to the CIM Object Manager
cc.createInstance(new CIMObjectPath(), ci);
```

▼ How to Register a Provider

You register a new or modified provider with the CIM Object Manager to communicate information about the data and operations the provider supports, and to notify the CIM Object Manager of the provider's location. The CIM Object Manager uses this information to load and initialize the provider, and to determine the appropriate provider for a particular client request.

1. **Create a Managed Object Format (MOF) file that defines the classes that the provider supports.**

Note – For more information on creating MOF files, see the DMTF Website at `http://www.dmtf.org`.

---

2. **Include the provider qualifier in the MOF file to specify the provider type and location for the CIM Object Manager.**

   For example:

```
[Provider("java:com.sun.providers.myprovider")]
Class_name {
...
};
```

   This qualifier indicates the following:

   - `java:` – The provider is written in the Java language and implements the `javax.wbem.provider` interfaces.
   - `com.sun.providers.myprovider` – The name of the Java class that implements the provider.

3. **Compile the MOF file by using the`mofcomp`(1M) command.**

EXAMPLE 4–4 Registering a Provider

This MOF file declares the `Ex_SimpleCIMInstanceProvider` class that is served by `SimpleCIMInstanceProvider`.

```
// ==========================================================
// Title:       SimpleCIMInstanceProvider
// Filename:    SimpleCIMInstanceProvider.mof
// Description:
// ==========================================================================

// ==========================================================================
// Pragmas
// ==========================================================================
#pragma Locale ("en-US")

// ==========================================================================
//   SimpleCIMInstanceProvider
// ==========================================================================
[Provider("java:SimpleCIMInstanceProvider")]
class Ex_SimpleCIMInstanceProvider
{
   // Properties
      [Key, Description("First Name of the User")]
   string First;
      [Description("Last Name of the User")]
   string Last;
};
```

CHAPTER 5

Writing WBEM Queries

This chapter explains how to use the WBEM Query Language (WQL) and the query APIs to write queries, and includes the following topics:

- "About the WBEM Query Language" on page 107
- "Writing Queries" on page 108
- "Parsing Queries" on page 111

Note – For detailed information on the WBEM query APIs (`javax.wbem.query`), see file:/usr/sadm/lib/wbem/doc/index.html.

About the WBEM Query Language

The WBEM Query Language (WQL) is a subset of the standard American National Standards Institute Structured Query Language (ANSI SQL) with semantic changes to support WBEM in the Solaris environment.

The following table shows the mapping of SQL concepts to WQL.

TABLE 5–1 Mapping of SQL Concepts to WQL

SQL Concept	WQL Representation
Table	CIM class
Row	CIM instance
Column	CIM property

Note – Like SQL, WQL statements use single (‘ ’) quotation marks.

In the Solaris WBEM Services implementation, WQL is a retrieval-only language. You can use WQL to query data that is stored using the CIM data model. In the CIM model, information about objects is stored in CIM classes and CIM instances. CIM instances can contain properties, which have a name, data type, and value.

Writing Queries

WBEM clients use WQL to query and filter data. When the data is served by a particular provider, the CIM Object Manager passes the client queries to the appropriate provider. You can search for instances that match a specified query in a particular class, or in all classes within a particular namespace.

For example, you can search for all instances of the `Solaris_DiskDrive` class that have a particular value for the `Storage_Capacity` property:

```
select * from Solaris_DiskDrive where Storage_Capacity = 1000
```

WQL Key Words

The Solaris WBEM SDK supports Level 1 WBEM SQL, which enables simple select operations without joins. The following table describes the supported WQL key words.

TABLE 5–2 Supported WQL Key Words

Key Word	Description
AND	Combines two Boolean expressions and returns TRUE when both expressions are TRUE.
FROM	Specifies the classes that contain the properties listed in a SELECT statement.
NOT	Comparison operator used with NULL.
OR	Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators.
SELECT	Specifies the properties that are used in a query.

TABLE 5–2 Supported WQL Key Words *(Continued)*

Key Word	Description
WHERE	Narrows the scope of a query.
LIKE	Generates a result set based on a minimum amount of information provided.

SELECT Statement

You use the SELECT statement to retrieve instances of a single class and its subclasses. You can also specify the properties to retrieve and the conditions that must be met.

Note – Currently, join operations are not supported.

The syntax for the SELECT statement is as follows:

`SELECT` *list* `FROM` *class* `WHERE` *condition*

The following table shows examples of using arguments in the SELECT clause to refine a search.

TABLE 5–3 Sample SELECT Statements

Example Query	Description
`SELECT * FROM` *class*	Selects all instances of the specified class and all of its subclasses. Each instance returned contains all the properties.
`SELECT PropertyA FROM` *class*	Selects all instances containing `PropertyA` of the specified class and all of its subclasses.
`SELECT PropertyA, PropertyB FROM` *class* `WHERE PropertyB=20`	Selects all instances of the specified class and all of its subclasses where `PropertyB=20`. Each returned instance contains only `PropertyA` and `PropertyB`.

FROM Clause

The FROM clause identifies the class in which to search for instances that match the query string. Only non-joined expressions are supported, which means that a valid WQL FROM clause includes only a single class.

The FROM clause is represented by the abstract class, `fromExp`. Currently `NonJoinExp` is the only direct subclass of `fromExp`. The `NonJoinExp` subclass represents FROM clauses with only one table (CIM class) to which the SELECT operation is applied.

WHERE Clause

The WHERE clause narrows the scope of a query. This clause contains a conditional expression, which can contain a property or key word, an operator, and a constant.

The syntax for a WHERE clause appended to a SELECT statement is as follows:

`SELECT` *CIMinstance* `FROM` *CIMclass* `WHERE` *conditional_expression*

The *conditional_expression* in the WHERE clause takes the following form:

property operator constant

The expression is composed of a property or key word, an operator, and a constant. You can append the WHERE clause to the SELECT statement using one of the following forms:

`SELECT` *instance* `FROM` *class* `[WHERE` *constant operator property*`]`

Valid WHERE clauses follow these rules:

- The value of the constant must be of the correct data type for the property.
- The operator must be a valid WQL operator.
- Either a property name or a constant must appear on either side of the operator in the WHERE clause.
- Arbitrary arithmetic expressions cannot be used. For example, the following query returns only instances of the `Solaris_Printer` class that represent a printer with ready status:

  ```
  SELECT * FROM Solaris_Printer WHERE Status = 'ready'
  ```

- Multiple groups of properties, operators, and constants can be combined in a WHERE clause using logical operators and parenthetical expressions. Each group must be joined with the AND, OR, or NOT operators.

  This example retrieves all instances of the `Solaris_FileSystem` class with the Name property set to either `home` or `files`:

  ```
  SELECT * FROM Solaris_FileSystem WHERE Name= 'home' OR Name= 'files'
  ```

  This example retrieves disks named `home` and `files` only if the disks have a certain amount of available space remaining, and have a Solaris file systes.

  ```
  SELECT * FROM Solaris_FileSystem WHERE (Name = 'home' OR
  Name = 'files') AND AvailableSpace > 2000000 AND FileSystem = 'Solaris'
  ```

Standard WQL Operators for WHERE Clauses

You can use the following standard WQL operators for a binary expression in the WHERE clause of a SELECT statement.

TABLE 5–4 WQL Operators for WHERE Clauses

Operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Parsing Queries

The `javax.wbem.query` package contains utility classes that you use to parse WQL queries. The main class is `SelectExp`, whose constructor takes in a WQL query string. `SelectExp` parses the string and splits it into three parts. These parts can be retrieved using their corresponding accessor methods, as shown in the following table.

Query Part	Accessor Method
SELECT list	`getSelectList`
FROM clause	`getFromClause`
WHERE clause	`getWhereClause`

The following query, onced parsed, has a SELECT list containing `PropertyA` and `PropertyB`. The FROM clause contains `test_class`, and the WHERE clause contains a parse tree of the conditional expression.

```
select PropertyA, PropertyB from test_class where
       PropertyA > 20 and PropertyB < 30
```

SELECT List

The select list returned by the `getSelectList` method for each `SelectExp` is an instance of the `SelectList` class. This list specifies the properties that must be included in the selected instances, and consists of a list of `AttributeExp` instances. You can retrieve these `AttributeExp` instances using the elements method of `SelectList`. Each attribute denotes the name of a column that maps to a property of

a CIMInstance in WQL. The AttributeExp has an apply method, which when passed in a CIMInstance, returns the value of the property that the AttributeExp represents. The SelectList has an apply method, which when passed in a CIMInstance, returns a CIMInstance containing only the properties that the SelectList AttributeExp instances denote.

FROM Clause

Currently, the only non-join expression that is allowed is the FROM clause. An instance of NonJoinExp is returned when the getFromClause method is invoked on SelectExp. The NonJoinExp represents the name of the class on which the selection is performed.

WHERE Clause

The WHERE clause is represented by QueryExp, an abstract class. The concrete subclasses are AndQueryExp, OrQueryExp, NotQueryExp, and BinaryRelQueryExp. Instances of these expressions are combined together in the form a parse tree that represents the original conditional expression.

The interior nodes of this tree consist of AndQueryExp, OrQueryExp, and NotQueryExp instances. These instances represent AND, OR, and NOT expressions, which in turn can consist of other AND, OR, and NOT expressions and binary relations.

The leaf nodes are BinaryRelQueryExp, which represent expressions of the form *property operator constant*. This represents a binary relation between a property and a constant value. You retrieve *property operator constant* using the getLeftValue, getRightValue, and getOperator methods.

Each QueryExp has an apply method, which when passed in a CIMInstance returns a boolean. The boolean is true if the conditional expression represented by the QueryExp is true for the CIMInstance. Otherwise, the boolean returns false.

The QueryExp has two other useful methods, canonizeDOC and canonizeCOD, which are used to simplify conditional expressions for further processing. The canonizeDOC method converts the parse tree from an arbitrary combination of ANDs and ORs to a canonical Disjunction of Conjunctions form (OR of ANDs). The canonizeCOD method converts the parse tree from an arbitrary combination of ANDs and ORs to canonical Conjunction of Disjunctions form (AND of ORs). These classes and methods are used by providers that need to filter instances based on input queries.

Note – More details of these classes can be found in the Javadoc reference pages. See `file:/usr/sadm/lib/wbem/doc/index.html`.

Writing a Provider That Handles Queries

Following is an example provider program that uses the query APIs to parse the WQL string passed to it by the `execQuery` method. This program parses the select expression in the query string, performs a deep enumeration of the class, and iterates through the instances in the enumeration, matching the query expression and select list to each instance. Finally, the program returns a vector containing the enumeration of the instances that match the query string.

EXAMPLE 5–1 Provider That Handles Queries

```
/*
 * The execQuery method will support only limited queries
 * based upon partial key matching. An empty Vector is
 * returned if no entries are selected by the query.
 *
 * @param  op The CIM object path of the CIM instance to be returned
 * @param  query The CIM query expression
 * @param  ql The CIM query language indicator
 * @param  cc The CIM class reference
 *
 * @return  A vector of CIM object instances
 *
 * @version    1.19 01/26/00
 * @author     Sun Microsystems, Inc.
 */
public CIMInstance[] execQuery(CIMObjectPath op,
                       String query,
                       String  ql,
                       CIMClass cc)
           throws CIMException {

   Vector result = new Vector();
   try {
       SelectExp q = new SelectExp(query);
       SelectList attrs = q.getSelectList();
       NonJoinExp from = (NonJoinExp)q.getFromClause();
       QueryExp where = q.getWhereClause();

       CIMInstance[] v = enumerateInstances(op, false, true,
                                            true, null, cc);

       // filtering the instances
       for (int i = 0; i < v.length; i++) {
               if ((where == null) || (where.apply(v[i]) == true)) {
                   result.addElement(attrs.apply(v[i]));
```

EXAMPLE 5–1 Provider That Handles Queries *(Continued)*

```
                }
            }
        } catch (Exception e) {
            throw new CIMException(CIMException.CIM_ERR_FAILED, e.toString());
        }
        return (CIMInstance[])result.toArray();
    } // execQuery
}
```

CHAPTER 6

Using the Solaris WBEM SDK Sample Programs

This chapter describes the sample programs provided with the Solaris WBEM SDK, and includes the following topics:

- "About the Sample Programs" on page 115
- "Sample Applet" on page 116
- "Sample Client Programs" on page 116
- "Sample Provider Programs" on page 118

About the Sample Programs

When you install the Solaris WBEM SDK, a sample Java applet and several programs are installed in `file:/usr/demo/wbem`. You can use these samples as a basis for developing your own applets and programs.

Note – To use the applet and sample programs, make sure that `/usr/java` points to at least JDK 1.2.2, and that the Solaris WBEM SDK files are installed in the `/usr` directory.

The following samples are provided:

- **Applet** – Lists and describes the Solaris software packages that are installed on a system running Solaris WBEM Services, and connects to the CIM Object Manager running on a local or a remote system.
- **Client programs** – Uses the Java APIs to make requests to the CIM Object Manager.
- **Provider programs** – Communicates with managed objects to access data.

Sample Applet

Note – For more detailed information on this applet, see: `file:/usr/demo/wbem/applet/README`.

You must run the applet on a machine that has network access to the CIM Object manager. In addition, the machine must run one of the following:

- JDK 1.2 Appletviewer
- Web browser that uses at least version 1.2.2 of the Java runtime environment, or has at least version 1.2.2 of the Java Plug-in product installed.

For more information on the JDK Appletviewer or the Java runtime environment, see `http://java.sun.com`. For more information on Java Plug-in, see the *Solaris Java Plug-in User's Guide*

▼ How to Run the Sample Applet Using Appletviewer

- **Type the following command:**

```
% appletviewer -JD \
java.security.policy=/usr/demo/wbem/applet/applet.policy \
/usr/demo/wbem/applet/GetPackageInfoAp.html
```

▼ How to Run the Sample Applet in a Web Browser

- **Open the following file in your web browser:**

  `file:/usr/demo/wbem/applet/GetPackageInfoAp.html`

Sample Client Programs

The sample client programs are located in subdirectories of `/usr/demo/wbem/client` and are described in the following table.

TABLE 6–1 Sample Client Programs

Directory	Program	Purpose
`./batching`	`./TestBatch` *host_name user_name password classname* `[rmi|http]`	Perform `enumerateInstanceName`, `getClass`, and `enumerateInstances` in a single batching call.
`./enumeration`	`./ClientEnum` *host username password classname* `[rmi|http]`	Enumerate classes and instances in the specified class in the default namespace `root\cimv2` on the specified host.
`./events`	`./Subscribe` *host username password classname*	Subscribe to lifecycle events for a specified class, print events that occur within one minute of the subscription, and then unsubscribe to the events.
`./logging`	`./CreateLog` *host root_username root_password* `[rmi|http]`	Create a log record on the specified host.
	`./ReadLog` *host root_username root_password* `[rmi|http]`	Read a log record on the specified host.
`./misc`	`./DeleteClass` *host classname root_username root_password* `[rmi|http]`	Delete the specified class in the default namespace `root\cimv2` on the specified host.
	`./DeleteInstances` *host classname root_username root_password* `[rmi|http]`	Delete instances of the specified class in the default namespace `root\cimv2` on the specified host.
`./namespace`	`./CreateNameSpace` *host parentNS childNS root_username root_password* `[rmi|http]`	Connect to the CIM Object Manager as the specified user, and create a namespace on the specified host.
	`./DeleteNameSpace` *host parentNS childNS root_username root_password* `[rmi|http]`	Delete the specified namespace on the specified host.
`./query`	`./ExampleQuery` *host username password* `[rmi|http]` *WQL_query*	Create a test class with sample instances, and perform queries on that class.
	`./TestQuery` *host username password* `[rmi|http]` *WQL_query*	Perform the specified WQL query.
`./systeminfo`	`./SystemInfo` *host username password* `[rmi|http]`	Display Solaris processor and system information for the specified host in a separate window.

Running the Sample Client Programs

You must first set the `CLASSPATH` to include the necessary `.jar` files before you run the client programs.

▼ How to Set the `CLASSPATH`

- **Do one of the following:**
  - Using the C shell, type:

    ```
    % setenv CLASSPATH .:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar
    :/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
    ```

  - Using the Bourne shell, type:

    ```
    % set CLASSPATH=.:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar
    :/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
    ```

▼ How to Run the Sample Client Programs

Most of the client sample programs accept an optional parameter that specifies the protocol that you want to use to connect to the CIM Object Manager. RMI is the default protocol.

- **Run the sample client programs using the following format:**

  ```
  % java program_name parameters
  ```

  For example, the following runs the `SystemInfo` program by connecting to *myhost* as the *root* user with the *secret* password using the HTTP protocol.

  ```
  % java SystemInfo myhost root secret http
  ```

Sample Provider Programs

The sample provider programs are located in subdirectories of `/usr/demo/wbem/provider` and are described in the following table.

TABLE 6–2 Sample Provider Programs

File Name	Purpose
`NativeProvider.java`	Top-level provider program that fulfills requests from the CIM Object Manager and routes them to the `Native_Example` provider. This program implements the `instanceProvider` and `methodProvider` APIs, and declares methods that enumerate instances and get an instance of the `Native_Example` class. This program also declares a method that invokes a method to print the string "Hello World."
`Native_Example.mof`	Creates a class that registers the `NativeProvider` provider with the CIM Object Manager. This MOF file identifies `NativeProvider` as the provider to service requests for dynamic data in the `Native_Example` class, and declares the properties and methods to be implemented by the `NativeProvider.`
`Native_Example.java`	The `NativeProvider` program calls this provider to implement methods that enumerate instances and get an instance of the `Native_Example` class. The `Native_Example` provider uses the APIs to enumerate objects and create instances of objects. The `Native_Example` class declares native methods, which call C functions in the `native.c` file to get system-specific values, such as host name, serial number, release, machine, architecture, and manufacturer.
`native.c`	This C program implements calls from the `Native_Example` Java provider in native C code.
`Native_Example.h`	A machine-generated header file for the `Native_Example` class. Defines the correspondence between the Java native method names and the native C functions that execute those methods.
`libnative.so`	Binary native C code compiled from the `native.c` file.

▼ How to Run the Sample Provider Programs

You must set up your environment before you can run the sample provider programs.

1. **Set the `LD_LIBRARY_PATH` environment variable to the location of the provider class files:**

   - Using the C shell, type:

     ```
     % setenv LD_LIBRARY_PATH /usr/sadm/lib/wbem
     ```

   - Using the Bourne shell, type:

     ```
     % LD_LIBRARY_PATH = /usr/sadm/lib/wbem
     ```

2. **Copy the `libnative.so` shared library file to the directory specified by the `LD_LIBRARY_PATH` environment variable:**

   ```
   % cp libnative.so /usr/sadm/lib/wbem
   ```

3. **Move the provider class files to the same path as the package in which they are defined:**

   ```
   % mv *.class /usr/sadm/lib/wbem
   ```

4. **Become root superuser.**

5. **Stop the CIM Object Manager in the same shell in which you set the `LD_LIBRARY_PATH` environment variable:**

   ```
   # /etc/init.d/init.wbem stop
   ```

   Note – When you set the `LD_LIBRARY_PATH` environment variable in a shell, you must stop and restart the CIM Object Manager in the same shell to recognize the new variable.

   ---

6. **Start the CIM Object Manager:**

   ```
   # /etc/init.d/init.wbem start
   ```

7. **Exit being superuser.**

8. **Compile the program's associated `.mof` file to load the appropriate class in the CIM Object Manager and to identify the provider:**

   ```
   % mofcomp -u root -p root_password Native_Example.mof
   ```

9. **Start CIM WorkShop:**

   ```
   % /usr/sadm/bin/cimworkshop
   ```

10. **In the CIM WorkShop Toolbar, click the Find Class icon.**

11. **In the Input dialog box, type the name of the class that you want to display and then click OK.**

    The class displays in CIM Workshop.

APPENDIX **A**

WBEM Error Messages

This appendix discusses the error messages generated by components of Solaris WBEM Services and the Solaris WBEM SDK, and covers the following topics:

- "About WBEM Error Messages" on page 121
- "List of Error Messages" on page 122

About WBEM Error Messages

The CIM Object Manager generates error messages that are used by both the Managed Object Format (MOF) Compiler and CIM WorkShop. The MOF Compiler appends a line to the error message that indicates the line number in which the error occurs in the `.mof` file.

Note – For more information on the MOF compiler, see `mofcomp`(1M).

Parts of an Error Message

An error message consists of the following parts:

- **Unique identifier** – A character string that identifies the error message. You can search for the unique identifier in the Javadoc reference pages to see an explanation of the content of the error message.
- **Parameters** – Placeholders for the specific classes, methods, and qualifiers that are cited in the exception message.

EXAMPLE A–1 Parts of an Error Message

The MOF compiler returns the following error:

EXAMPLE A–1 Parts of an Error Message *(Continued)*

```
REF_REQUIRED = Association class CIM_Docked needs at least two refs.
Error in line 12.
```

- `REF_REQUIRED` is the *unique identifier*.
- `CIM_Docked` is a *parameter*.
- `line 12` indicates the *line number* in the `.mof` file in which the error occurred.

List of Error Messages

This section describes the WBEM error messages, sorted by unique identifier.

`ABSTRACT_INSTANCE`

Description: This error message uses one parameter, `{0}`, which is replaced by the name of the abstract class.

Cause: A create instance was attempted for the instance. However, the specified class is an abstract class, and abstract classes cannot have instances.

Solution: Create instances for concrete classes.

`CANNOT_ASSUME_ROLE`

Description: This error message uses two parameters:

- `{0}` is replaced by the user name.
- `{1}` is replaced by the role name.

Cause: The specified principal cannot assume the specified role.

Solution: Make sure that the user has the appropriate rights to assume the given role. If the user does not have the appropriate rights, contact your system administrator.

`CHECKSUM_ERROR`

Description: This error message does not use parameters.

Cause: The message could not be sent because it was damaged or corrupted. The damage could have occurred accidentally in transit or by a malicious third party.

Note – This error message is displayed when the CIM Object Manager receives an invalid checksum. A checksum is the number of bits in a packet of data passed over the network. This number is used by the sender and the receiver of the information to ensure that the transmission is secure and that the data has not been corrupted or intentionally modified during transit.

An algorithm is run on the data before transmission. Then the checksum is generated and included with the data to indicate the size of the data packet. When the message is received, the receiver can recompute the checksum and compare it to the sender's checksum. If the checksums match, the transmission was secure and the data was not corrupted or modified.

Solution: Resend the message using Solaris WBEM Services security features. For information about Solaris WBEM Services security, see "Administering Security" in the *Solaris WBEM Services Administration Guide*.

`CIM_ERR_ACCESS_DENIED`

Description: This error message does not use parameters.

Cause: This error message is displayed when a user does not have the appropriate privileges and permissions to complete an action.

Solution: See your system administrator or the person who is responsible for your CIM Object Manager to request privileges to complete the operation.

`CIM_ERR_ALREADY_EXISTS`

Instance 1: CIM_ERR_ALREADY_EXISTS

Description: This instance uses one parameter, `{0}`, which is replaced by the name of the duplicate class.

Cause: The class you attempted to create uses the same name as an existing class.

Solution: In CIM WorkShop, search for existing classes to see the class names that are in use. Then create the class by using a unique class name.

Instance 2: CIM_ERR_ALREADY_EXISTS

Description: This instance uses one parameter, `{0}`, which is replaced by the name of the duplicate instance.

Cause: The instance for a class you attempted to create uses the same name as an existing instance.

Solution: In CIM WorkShop, search for existing instances to see the names that are in use. Then create the instance by using a unique name.

Instance 3: CIM_ERR_ALREADY_EXISTS

Description: This instance uses one parameter, `{0}`, which is replaced by the name of the duplicate namespace.

Cause: The namespace you attempted to create uses the same name as an existing namespace.

Solution: In CIM WorkShop, search for existing namespaces to see the names that are in use. Then create the namespace by using a unique name.

Instance 4: CIM_ERR_ALREADY_EXISTS

Description: This instance uses one parameter, `{0}`, which is replaced by the name of the duplicate qualifier type.

Cause: The qualifier type you attempted to create uses the same name as an existing qualifier type of the property it modifies.

Solution: In CIM WorkShop, search for qualifier types that exist for the property to see the names that are in use. Then create the qualifier type by using a unique name.

`CIM_ERR_CLASS_HAS_CHILDREN`

Description: This error message uses one parameter, `{0}`, which is replaced by the class name.

Cause: This exception is thrown by the CIM Object Manager to disallow invalidation of the subclasses by a super class deletion. Clients must explicitly delete the subclasses first. The check for subclasses is made before the check for class instances.

Solution: Remove the subclasses of the given class.

`CIM_ERR_CLASS_HAS_INSTANCES`

Description: This error message uses one parameter, `{0}`, which is replaced by the class name.

Cause: This exception is thrown if you attempt to delete a class that has instances.

Solution: Remove the instances of the given class.

`CIM_ERR_FAILED`

Description: This error message uses one parameter, `{0}`, which is replaced by a message that explains the error condition and its possible cause.

Cause: This error message is generic and can be displayed for many different error conditions.

Solution: The solution varies depending on the error condition.

CIM_ERR_INVALID_PARAMETER

Description: This error message uses one parameter, {0}, which is replaced by the name of the invalid parameter.

Cause: The name of the parameter or the method is invalid.

Solution: Fix the parameter.

CIM_ERR_INVALID_QUERY

Description: This error message uses two parameters:

- {0} is replaced by the invalid part of the query.
- {1} is replaced by additional information, including the actual error in the query.

Cause: The given query either has syntactical errors or semantic errors.

Solution: Fix the errors based on the exception details. In addition, make sure that the query string and query language match.

CIM_ERR_INVALID_SUPERCLASS

Description: This error message uses two parameters:

- {0} is replaced by the name of the specified subclass.
- {1} is replaced by the name of the class for which a specified subclass does not exist.

Cause: A class is specified to belong to a subclass from a superclass, but the superclass does not exist. The specified superclass might be misspelled, or a non-existent superclass name might have been specified in place of the intended superclass name. Or, the superclass and the subclass might have been interpolated. That is, the specified superclass may be a subclass of the subclass. In the previous example, CIM_Chassis is specified as the superclass of CIM_Container, but CIM_Chassis is a subclass of CIM_Container.

Solution: Check the spelling and the name of the superclass to ensure that it is correct. Ensure that the superclass exists in the namespace.

CIM_ERR_LOW_ON_MEMORY

Description: This error message does not use parameters.

Cause: The CIM Object Manager is low on memory.

Solution: Delete some class definitions and static instances to free up memory.

CIM_ERR_NOT_FOUND

Instance 1: CIM_ERR_NOT_FOUND

Description: This instance uses one parameter, {0}, which is replaced by the name of the non-existent class.

Cause: A class is specified, but it does not exist. The specified class might be misspelled, or a non-existent class name might have been accidentally specified in place of the intended class name.

Solution: Check the spelling and the name of the class to ensure that it is correct. Ensure that the class exists in the namespace.

Instance 2: CIM_ERR_NOT_FOUND

Description: This instance uses two parameters:

- `{0}` is replaced by the name of the specified instance.
- `{1}` is replaced by the name of the specified class.

Cause: The instance does not exist.

Solution: Create the instance.

Instance 3: CIM_ERR_NOT_FOUND

Description: This instance uses one parameter, `{0}`, the name of the specified namespace.

Cause: The specified namespace is not found. This error can occur if the name of the namespace was entered incorrectly due to a typing error or spelling mistake.

Solution: Retype the name of the namespace. Ensure that you type and spell the namespace correctly.

`CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED`

Description: This error message uses one parameter, `{0}`, which is replaced by the invalid query language string.

Cause: The requested query language is not recognized by CIM.

Solution: Provide a supported query language.

`CLASS_REFERENCE`

Description: This error message uses two parameters:

- `{0}` parameter is replaced by the name of the class that was defined to participate in a reference.
- `{1}` parameter is replaced by the name of the reference.

Cause: A property was defined for a class to indicate that the class has a reference. However, the class is not an association. A class can only have a reference as a property if it is an association.

Solution: Add the association qualifier or remove the reference.

INVALID_DATA

Description: This error message does not use parameters.

Cause: The security authenticator data is invalid or is not consistent with the security mechanism you are using.

Solution: Make sure that your security modules are configured correctly.

INVALID_CREDENTIAL

Description: This error message does not use parameters.

Cause: This error message is displayed when you enter an invalid password, or if your CLASSPATH is not set up to include authentication checks for client applications.

Solution:

- Use the correct password.
- Make sure your CLASSPATH contains the following: `/usr/sadm/lib/wbem/extension:/usr/sadm/lib/wbem/sunwbem.jar`

INVALID_QUALIFIER_NAME

Description: This error message uses one parameter, `{0}`, which is replaced by the MOF notation that depicts an empty qualifier name.

Cause: A qualifier was created for a property, but a qualifier name was not specified.

Solution: Include the qualifier name.

KEY_OVERRIDE

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the nonabstract class that is in an override relationship with a class that has one or more key qualifiers.
- `{1}` is replaced by the name of the concrete class that has the key qualifier.

Cause: A nonabstract class, referred to as a concrete class, is put into an override relationship with a concrete class that has one or more key qualifiers. In CIM, all concrete classes require at least one key qualifier, and a non-key class cannot override a class that has a key.

Solution: Create a key qualifier for the non-key class.

KEY_REQUIRED

Description: This error message uses one parameter, `{0}`, which is replaced by the name of the class that requires a key.

Cause: A key qualifier was not provided for a concrete class. In CIM, all non-abstract classes, referred to as concrete classes, require at least one key qualifier.

Solution: Create a key qualifier for the class.

`METHOD_OVERRIDDEN`

Description: This error message uses three parameters:

- `{0}` is replaced by the name of the method that is trying to override the method represented by parameter `{1}`.
- `{1}` is replaced by the name of the method that has already been overridden by the method represented by parameter `{2}`.
- `{2}` is replaced by the name of the method that has overridden parameter `{1}`.

Cause: A method is specified to override another method that has already been overridden by a third method. Once a method has been overridden, it cannot be overridden again.

Solution: Specify a different method to override.

`NEW_KEY`

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the key.
- `{1}` is replaced by the name of the class that is trying to define a new key.

Cause: A class is trying to define a new key when keys already have been defined in a superclass. Once keys have been defined in a superclass, new keys cannot be introduced into the subclasses.

Solution: Do not define a new key.

`NO_CIMOM`

Description: This error message uses one parameter, `{0}`, which is replaced by the name of the host that is expected to be running the CIM Object Manager.

Cause: The CIM Object Manager is not running on the specified host.

Solution: Ensure that the CIM Object Manager is running on the host to which you are trying to connect. If the CIM Object Manager is not running on that host, connect to a host running the CIM Object Manager.

`NO_EVENT_PROVIDER`

Description: An event provider cannot be found.

Cause: The property provider class is not found.

Solution: Ensure that the class path of the CIM Object Manager contains the provider class parameters, the indication class for which the provider is being defined, and the name of the Java provider class. Ensure that the CIM Object Manager Solaris provider is set and the provider qualifier is correct.

`NOT_EVENT_PROVIDER`

Description: This error message does not use parameters.

Cause: The provider class present in the class path does not implement the `EventProvider` interface.

Solution: Ensure that the provider is correct and register the provider implements.

`NO_INSTANCE_PROVIDER`

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the class for which the instance provider cannot be found.
- `{1}` is replaced by the name of the instance provider class that was specified.

Cause: The Java class of the specified instance provider is not found. This error message indicates that the class path of the CIM Object Manager is missing one or more of the following:

- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

Solution: Set the CIM Object Manager environment variable. Ensure that the CIM Object Manager Solaris provider is set and the provider qualifier is correct.

`NO_METHOD_PROVIDER`

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the class for which the method provider cannot be found.
- `{1}` is replaced by the name of the method provider class that was specified.

Cause: The Java class of the specified method provider is not found. This error message indicates that the class path of the CIM Object Manager is missing one or more of the following:

- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

Solution: Set the CIM Object Manager class path. Ensure that the CIM Object Manager Solaris Provider is set and the provider qualifier is correct.

NO_OVERRIDDEN_METHOD

Description: This error message uses two parameters:

- {0} is replaced by the name of the method that has overridden the method represented by {1}.
- {1} is replaced by the name of the method that has been overridden.

Cause: The method of a subclass is trying to override the method of the superclass. However, the method that you are trying to override does not exist in the class hierarchy because it has not been defined.

When you override a method, you override its implementation and its signature.

Solution: Ensure that the method exists in the superclass.

NO_OVERRIDDEN_PROPERTY

Description: This error message uses two parameters:

- {0} is replaced by the name of the property that has overridden {1}.
- {1} is replaced by the name of the overriding property.

Cause: The property of a subclass is trying to override the property of the superclass. However, the property that you are trying to override does not exist in the class hierarchy because it has not been defined.

Solution: Ensure that the property exists in the superclass.

NO_PROPERTY_PROVIDER

Description: This error message uses two parameters:

- {0} is replaced by the name of the class for which the property provider cannot be found.
- {1} is replaced by the name of the property provider class that was specified.

Cause: The Java class of the specified property provider is not found. This error message indicates that the class path of the CIM Object Manager is missing one or more of the following:

- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

Solution: Set the CIM Object Manager class path. Ensue that the CIM Object Manager is running on the host to which you are trying to connect. If the CIM Object Manager is not running on that host, connect to a host running the CIM Object Manager.

NO_QUALIFIER_VALUE

Description: This error message uses two parameters:

- {0} is replaced by the name of the qualifier that modifies the element {1}.
- {1} is the element to which the qualifier refers. Depending on the qualifier, {1} can be a class, property, method, or reference.

Cause: A qualifier was specified for a property or method, but values were not included for the qualifier. For example, the qualifier VALUES requires a string array to be specified. If the VALUES qualifier is specified without the required string array, the NO_QUALIFIER_VALUE error message is displayed.

Solution: Specify the required parameters for the qualifier. For information about what attributes are required for which qualifiers, see the DMTF CIM Specification at http://www.dmtf.org.

NO_SUCH_METHOD

Description: This error message uses two parameters:

- {0} is replaced by the name of the specified method.
- {1} is replaced by the name of the specified class.

Cause: Most likely, the method was not defined for the specified class. If the method is defined for the specified class, another method name might have been misspelled or typed differently in the definition.

Solution: Define the method for the specified class. Ensure that the method name and the class name are spelled correctly.

NO_SUCH_PRINCIPAL

Description: This error message uses one parameter, {0}, which is replaced by the name of the principal, a user account.

Cause: The specified user account cannot be found. The user name might have been misspelled, or a the user does not have a user account.

Solution: Ensure that the user name is spelled and typed correctly upon login. Ensure that the user has a user account.

NO_SUCH_QUALIFIER1

Description: This error message uses one parameter, {0}, which is replaced by the name of the undefined qualifier.

Cause: A new qualifier was specified but it was not defined as part of the extension schema. The qualifier must be defined as part of the CIM Schema or part of an extension schema to be recognized as a valid qualifier for a property or method of a particular class.

Solution: Define the qualifier as part of the extension schema, or use a standard CIM qualifier. For information about standard CIM qualifiers and the usage of qualifiers in the CIM Schema, see the DMTF CIM Specification at: `http://www.dmtf.org`.

`NO_SUCH_QUALIFIER2`

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the class, property, or method that the qualifier modifies.
- `{1}` is replaced by the name of the qualifier that cannot be found.

Cause: A new qualifier was specified to modify a property or method of a particular class. The qualifier was not defined as part of the extension schema. The qualifier must be defined as part of the CIM Schema or part of an extension schema to be recognized as a valid qualifier for a property or method of a particular class.

Solution: Define the qualifier as part of the extension schema or use a standard CIM qualifier. For information about standard CIM qualifiers and the usage of qualifiers in the CIM schema, see the DMTF CIM Specification at: `http://www.dmtf.org`.

`NO_SUCH_ROLE`

Description: This error message uses one parameter, `{0}`, which is replaced by the role name.

Cause: The specified role cannot be found or is not a role identity.

Solution: Make sure that the input role exists. If the role is required, contact your system administrator to set up the role.

`NO_SUCH_SESSION`

Description: This error message uses one parameter, `{0}`, which is replaced by the session identifier.

Cause: The session has been closed and is being used subsequently.

Solution: Do not close the session.

`NOT_HELLO`

Description: This error message does not use parameters.

Cause: This error message is displayed if the data in the hello message, the first message sent to the CIM Object Manager, is corrupted.

Solution: No action is available in response to this error message. For information about Solaris WBEM Services security features, see "Administering Security" in the *Solaris WBEM Services Administration Guide*.

NOT_INSTANCE_PROVIDER

Description: This error message uses two parameters:

- {0} is replaced by the name of the instance for which the InstanceProvider interface is being defined.
- {1} is replaced by the name of the Java provider class that does not implement the InstanceProvider interface. The InstanceProvider interface must be implemented to enumerate all instances of the specified class.

Cause: The path to the Java provider class specified by the CLASSPATH environment variable does not implement the InstanceProvider interface.

Solution: Ensure that the provider is correct and register the provider implements.

NOT_METHOD_PROVIDER

Description: This error message uses two parameters:

- {0} is replaced by the name of the method for which the MethodProvider interface is being defined. The MethodProvider interface causes a specified method to be implemented in a program and enacted.
- {1} is replaced by the name of the Java provider class that does not implement the MethodProvider interface.

Cause: The Java provider class present in the class path does not implement the MethodProvider interface.

Solution: Ensure that the Java provider class present in the class path implements the MethodProvider interface. Use the following command when you declare the provider: `public Solaris implements MethodProvider`.

NOT_PROPERTY_PROVIDER

Description: This error message uses two parameters:

- {0} is replaced by the name of the method for which the PropertyProvider interface is being defined. The PropertyProvider interface is required to retrieve the values of the specified property.
- {1} is replaced by the name of the Java provider class that does not implement the PropertyProvider interface.

Cause: The Java provider class present in the class path does not implement the PropertyProvider interface.

Solution: Ensure that the Java provider class present in the class path implements the PropertyProvider interface. Use the following command when you declare the provider: `public Solaris implements PropertyProvider`.

NOT_RESPONSE

Description: This error message does not use parameters.

Cause: This error message is displayed when the data in a first response message from the CIM Object Manager is corrupted.

Solution: No action is available in response to this error message. For information about Solaris WBEM Services security features, see "Administering Security" in the *Solaris WBEM Services Administration Guide*.

`PROPERTY_OVERRIDDEN`

Description: This error message uses three parameters:

- `{0}` is replaced by the name of the property that is trying to override the property represented by parameter `{1}`.
- `{1}` is replaced by the name of the property that already has been overridden.
- `{2}` is replaced by the name of the property that has overridden the property represented by parameter `{1}`.

Cause: A property is specified to override another method that has already been overridden by a third method. Once a property has been overridden, it cannot be overridden again.

Solution: Specify a different property to override.

`QUALIFIER_UNOVERRIDABLE`

Description: This error message uses two parameters:

- `{0}` is replaced by the name of the qualifier with the `DisableOverride` flavor.
- `{1}` is replaced by the name of the qualifier that is disabled by `{0}`.

Cause: The qualifier that is being overridden has the `DisableOverride` flavor.

Solution: Reset the ability of the qualifier to `EnableOverride` or to `Override=True`.

`REF_REQUIRED`

Description: This error message uses one parameter, `{0}`, which is replaced by the name of the class.

Cause: A class was defined to participate in an association, but no references were cited. The rules of the Common Information Model specify that an association must contain one or more references.

Solution: Add references or remove the association qualifier.

`SCOPE_ERROR`

Description: This error message uses three parameters:

- `{0}` is replaced by the name of the class the specified qualifier modifies.
- `{1}` is replaced by the name of the specified qualifier.

- {2} is replaced by the type of attribute that the qualifier modifies.

Cause: A qualifier was specified in a manner that conflicts with the qualifier type definition. The scope of the [READ] qualifier is the definition that directs the [READ] qualifier to modify a property. If the [READ] qualifier is used in a manner other than the direction of its scope, for example, if the [READ] qualifier is specified to modify a method, the SCOPE_ERROR message is returned.

Note – The Common Information Model (CIM) Specification defines the types of CIM elements that a CIM qualifier can modify. This definition of the way in which a qualifier can be used is referred to as its scope. Most qualifiers, by definition, have a scope that directs them to modify properties or methods or both. Many qualifiers have a scope that directs them to modify parameters, classes, associations, indications, or schemas.

Solution: Confirm the scope of the specified qualifier. Refer to the Qualifiers section of the DMTF CIM Specification at:http://www.dmtf.org for the standard definitions of CIM qualifiers. Use a different qualifier for the results you want to achieve, or change your program to use the qualifier according to its CIM definition.

TYPE_ERROR

Description: This error message uses five parameters:

- {0} is replaced by the name of the specified element, such as a property, method, or qualifier.
- {1} is replaced by the name of the class to which the specified element belongs.
- {2} is replaced by the type defined for the element.
- {3} is replaced by the type of value assigned.
- {4} is replaced by the actual value assigned.

Cause: The value of a property parameter or method parameter and its defined type are mismatched.

Solution: Match the value of the property or method with its defined type.

UNKNOWNHOST

Description: This error message uses one parameter, {0}, which is replaced by the name of the host.

Cause: A call was made to a specified host. The specified host is unavailable or cannot be located. It is possible that the host name was misspelled, the host computer was moved to a different domain, the host name has not been registered in the list of hosts that belong to the domain, or the host is temporarily unavailable due to system conditions.

Solution: Check the spelling of the host name. Use the `ping` command to ensure that the host computer is responding. Check the system conditions of the host. Ensure that the host belongs to the specified domain.

`VER_ERROR`

Description: This error message uses one parameter, `{0}`, which is replaced by the version number of the running CIM Object Manager.

Cause: The CIM Object Manager does not support the version of the client that is trying to connect to it.

Solution: Install the appropriate version.

APPENDIX **B**

The Solaris Schema

The Solaris Schema and CIM Schema are available by default in the CIM Object Manager. You can view the MOF files, from which the Solaris Schema and CIM Schema are compiled, in `/usr/sadm/mof/`. CIM Schema files, which implement the Core and Common models of the Common Information Model, are denoted by the use of "CIM" in their associated file names. The Solaris Schema files, denoted by the use of "Solaris" in their file names, provide the unique extensions that Sun Microsystems has made to the Common Information Model.

Documentation of the Solaris providers listed in this chapter is included in the MOF file in which the provider is specified.

- "`Solaris_Schema1.0.mof` File" on page 139
- "`Solaris_CIMOM1.0.mof` File" on page 140
- "`Solaris_Core1.0.mof` File" on page 140
- "`Solaris_Application1.0.mof` File" on page 140
- "`Solaris_System1.0.mof` File" on page 141
- "`Solaris_Device1.0.mof` File" on page 142
- "`Solaris_Acl1.0.mof` File" on page 143
- "`Solaris_Network1.0.mof` File" on page 143
- "`Solaris_Users1.0.mof` File" on page 143
- "`Solaris_Event1.0.mof` File" on page 143
- "`Solaris_SNMP1.0.mof` File" on page 144
- "`Solaris_VM1.0.mof` File" on page 144
- "`Solaris_Project1.0.mof` File" on page 145
- "`Solaris_Performance1.0.mof` File" on page 146

Solaris Schema Files

This table provides a brief overview of the Solaris Schema files in `/usr/sadm/mof`.

TABLE B–1 Solaris Schema Files

Solaris Schema File	What This Schema File Provides
`Solaris_Schema1.0.mof`	Lists all of the MOF files of the Solaris Schema, in `#pragma Include` statements. Specifies the order in which the MOF files are read and compiled.
`Solaris_CIMOM1.0.mof`	Contains configuration information for the CIM Object Manager.
`Solaris_Core1.0.mof`	Enables WBEM core features to be implemented. Enables you to set locales, qualifiers, and providers.
`Solaris_Application1.0.mof`	Models Solaris packages and patches in CIM.
`Solaris_System1.0.mof`	Models the Solaris Schema components for a system, including the operating system and processes of the system. Extends CIM Schema definitions through the definition of the `Solaris_Process` and `Solaris_OperatingSystem` classes.
`Solaris_Device1.0.mof`	Enables a description of your system's processor, serial ports, printing devices, and time settings to make your computer work with the CIM Object Manager.
`Solaris_Acl1.0.mof`	Contains the classes for WBEM access control list (ACL) based security.
`Solaris_Network1.0.mof`	Defines classes pertaining to network domains, IP subnets, and naming services (including NIS, NIS+, LDAP, DNS, and server `/etc` files).
`Solaris_Users1.0.mof`	Defines classes for working with user accounts.
`Solaris_Event1.0.mof`	Defines unique Solaris indication handlers. The class that is defined in this file facilitates the delivery of indications to management clients over Sun Microsystems' implementation of the CIM Remote Method Invocation (RMI) protocol.
`Solaris_SNMP1.0.mof`	Defines classes that pertain to configuration information for an SNMP device.
`Solaris_VM1.0.mof`	Defines classes that pertain to storage devices.

TABLE B–1 Solaris Schema Files *(Continued)*

Solaris Schema File	What This Schema File Provides
`Solaris_Project1.0.mof`	Defines classes that model the Solaris project database.
`Solaris_Performance1.0.mof`	Defines classes that pertain to computing resource metrics, that is, classes that pertain to the use and performance of computing resources for each user and for each project.

The following sections describe the contents of each schema file in more detail.

`Solaris_Schema1.0.mof` File

The `Solaris_Schema1.0.mof` file is the high-level container of all other MOF files that make up the Solaris Schema. This file lists the MOF files in the order in which you must compile them.

The Java classes that you generate from each compilation are then sent to the CIM Object Manager, where they are either enacted as events or sent to the CIM Object Manager Repository for storage as objects. The following listing of the `Solaris_Schema1.0.mof` file shows the `Include` statements in the order that is required for compilation.

```
/*
Title        Solaris Master MOF 1.0
Description include pragmas for all other mofs
Date        03/10/01
Version     1.1
Copyright   (c) 2000 Sun Microsystems, Inc. All Rights Reserved.
*/

#pragma Include ("Solaris_Core1.0.mof")
#pragma Include ("Solaris_Application1.0.mof")
#pragma Include ("Solaris_System1.0.mof")
#pragma Include ("Solaris_Device1.0.mof")
#pragma Include ("Solaris_Network1.0.mof")
#pragma Include ("Solaris_Users1.0.mof")
#pragma Include ("Solaris_Project1.0.mof")
#pragma Include ("Solaris_Event1.0.mof")
#pragma Include ("Solaris_CIMOM1.0.mof")
#pragma Include ("Solaris_SNMP1.0.mof")

// This must be the last include since it changes the CIM namespace
#pragma Include ("Solaris_Acl1.0.mof")
```

The compiler parses a line of the `Solaris_Schema1.0.mof` file, compiles the file specified in the `Include` statement, and then parses the next line of the `Solaris_Schema1.0.mof` file, until all included files are compiled.

Solaris_CIMOM1.0.mof File

The Solaris_CIMOM1.0.mof file contains all the system properties used by the CIM Object Manager.

The Solaris_CIMOM1.0.mof file defines the following classes:

- CIM_ObjectManager
- CIM_ObjectManagerCommunicationMechanism
- CIM_WBEMCommunicationMechanism
- Solaris_CIMOM
- Solaris_ObjectManagerClientProtocolAdapter
- Solaris_ObjectManagerProtocolAdapter
- Solaris_ObjectManagerProviderProtocolAdapter
- Solaris_ProviderPath

In addition, the Solaris_CIMOM1.0.mof file defines the association class CIM_CommMechanismForManager.

Solaris_Core1.0.mof File

The Solaris_Core1.0.mof file is the first of the Solaris Schema files to be compiled after the Solaris_Schema1.0.mof file. This file provides the definition of the Solaris_ComputerSystem class of the Solaris provider.

The Solaris_Core1.0.mof file defines the following classes:

- Solaris_ComputerSystem
- Solaris_LogRecord
- Solaris_LogService
- Solaris_Product
- Solaris_SystemDownStatisticalInformation
- Solaris_SystemUpStatisticalInformation

In addition, the Solaris_Core1.0.mof file defines the following association classes:

- Solaris_ProductParentChild
- Solaris_ProductProductDependency
- Solaris_SystemSetting

Solaris_Application1.0.mof File

The Solaris_Application1.0.mof file enables you to set up packages and patches for your applications that extend the Solaris Schema.

The Solaris_Application1.0.mof file defines the following classes:

- Solaris_InstalledSoftwareElement

- Solaris_Package
- Solaris_Patch
- Solaris_RegistrySoftwareElement
- Solaris_SoftwareElement
- Solaris_SoftwareFeature

In addition, the Solaris_Application1.0.mof file defines the following association classes:

- Solaris_PatchPackageDependency
- Solaris_PatchToPatchDependency
- Solaris_ProductSoftwareElementDependency
- Solaris_ProductSoftwareElements
- Solaris_ProductSoftwareFeatureDependency
- Solaris_ProductSoftwareFeatures
- Solaris_RegistryElementDependency
- Solaris_SoftwareElementDependency
- Solaris_SoftwareElementProductDependency
- Solaris_SoftwareElementSoftwareFeatureDependency
- Solaris_SoftwareFeatureDependency
- Solaris_SoftwareFeatureParentChild
- Solaris_SoftwareFeatureProductDependency
- Solaris_SoftwareFeatureSoftwareElementDependency
- Solaris_SoftwareFeatureSoftwareElements

Solaris_System1.0.mof File

The Solaris_System1.0.mof file defines the following classes:

- Solaris_CpuSysinfo
- Solaris_CpuUtilizationInformation
- Solaris_CpuVminfo
- Solaris_DataFile
- Solaris_DiskIOInformation
- Solaris_DisklessClient
- Solaris_Eeprom
- Solaris_EepromSetting
- Solaris_InstalledOS
- Solaris_JobScheduler
- Solaris_JobScheduler_Cron
- Solaris_OperatingSystem
- Solaris_OSProcess
- Solaris_OsService
- Solaris_Process
- Solaris_RunningOS
- Solaris_ScheduledJob
- Solaris_ScheduledJob_Cron

In addition, the Solaris_System1.0.mof file defines the following association classes:

- Solaris_EepromElementSetting
- Solaris_HostedJobScheduler
- Solaris_OwningJobScheduler
- Solaris_SystemDevice

Solaris_Device1.0.mof File

The Solaris_Device1.0.mof file defines the following classes:

- Solaris_Environment
- Solaris_EthernetAdapter
- Solaris_Keyboard
- Solaris_LogEntry
- Solaris_LogServiceProperties
- Solaris_LogServiceSetting
- Solaris_MessageLog
- Solaris_MessageLogRecord
- Solaris_MessageLogSetting
- Solaris_Printer
- Solaris_PrintJob
- Solaris_PrintQueue
- Solaris_PrintSAP
- Solaris_PrintService
- Solaris_Processor
- Solaris_SerialPort
- Solaris_SerialPortConfiguration
- Solaris_SerialPortSetting
- Solaris_SoundDevice
- Solaris_SyslogRecord
- Solaris_TimeZone

In addition, the Solaris_Device1.0.mof file defines the following association classes:

- Solaris_CpuSysinfoPerformanceMonitor
- Solaris_CpuUtilizationPerformanceMonitor
- Solaris_CpuVminfoPerformanceMonitor
- Solaris_LogInDataFile
- Solaris_OwningPrintQueue
- Solaris_PrinterServicingQueue
- Solaris_QueueForPrintService
- Solaris_RecordInLog
- Solaris_SystemTimeZone

Solaris_Acl1.0.mof File

The Solaris_Acl1.0.mof file specifies the Solaris WBEM Services security classes. This file defines these base classes for access control lists, users, and namespaces:

- Solaris_Acl
- Solaris_NamespaceAcl
- Solaris_UserAcl

Solaris_Network1.0.mof File

The Solaris_Network1.0.mof file defines classes that pertain to network domains, IP subnets, and naming services (including NIS, NIS+, LDAP, DNS, and server /etc files). The Solaris_Network1.0.mof file defines the following classes:

- Solaris_AdminDomain
- Solaris_DnsAdminDomain
- Solaris_IPProtocolEndpoint
- Solaris_IPSubnet
- Solaris_LdapAdminDomain
- Solaris_NisAdminDomain
- Solaris_NisplusAdminDomain
- Solaris_SystemAdminDomain

Solaris_Users1.0.mof File

The Solaris_Users1.0.mof file defines the following classes:

- Solaris_AuthorizationAttribute
- Solaris_EmailAlias
- Solaris_ExecutionProfile
- Solaris_MailBox
- Solaris_ProfileAttribute
- Solaris_ShellSAP
- Solaris_UserAccount
- Solaris_UserGroup
- Solaris_UserHomeDirectory
- Solaris_UserTemplate

Solaris_Event1.0.mof File

The Solaris_Event1.0.mof file contains classes that deal with indication handlers that are unique to Solaris. These Solaris indication handlers are subclasses of CIM_IndicationHandler. These subclasses include Solaris_RMIDelivery and

Solaris_JAVAXRMIDelivery. The client RMI protocol uses the Solaris_JAVAXRMIDelivery handler. Solaris_Event1.0.mof contains Solaris_RMIDelivery to ensure compatibility with previous versions of WBEM.

Solaris_SNMP1.0.mof File

The Solaris_SNMP1.0.mof file defines classes that pertain to configuration information for an SNMP device. The Solaris_SNMP1.0.mof file defines the following classes:

- Solaris_SNMPGroupConf
- Solaris_SNMPSystem
- Solaris_SNMPSystemConf

Solaris_VM1.0.mof File

The Solaris_VM1.0.mof file defines classes that pertain to storage devices, such as these:

- State database replicas within a slice
- Range of extents within a storage extent that can be used for data
- Stripes
- Concatenated stripes
- Mirrors
- RAID Level 5 devices
- UFS logging file systems
- Spare pools
- Disk sets
- Storage volumes

The Solaris_VM1.0.mof file defines the following classes:

- Solaris_Directory
- Solaris_DiskDrive
- Solaris_DiskPartition
- Solaris_HSFS
- Solaris_LocalFileSystem
- Solaris_MediaPresent
- Solaris_NFS
- Solaris_UFS
- Solaris_VMConcat
- Solaris_VMDiskSet
- Solaris_VMExtent
- Solaris_VMHotSparePool
- Solaris_VMMirror
- Solaris_VMRaid5

- Solaris_VMSoftPartition
- Solaris_VMStateDatabase
- Solaris_VMStorageVolume
- Solaris_VMStripe
- Solaris_VMTrans

In addition, the Solaris_VM1.0.mof file defines the following association classes:

- Solaris_DiskIOPerformanceMonitor
- Solaris_HSFSMount
- Solaris_Mount
- Solaris_NFSExport
- Solaris_NFSMount
- Solaris_UFSMount
- Solaris_VMConcatComponent
- Solaris_VMDriveInDiskSet
- Solaris_VMExtentBasedOn
- Solaris_VMExtentInDiskSet
- Solaris_VMHostInDiskSet
- Solaris_VMHotSpareInUse
- Solaris_VMHotSpares
- Solaris_VMMirrorSubmirrors
- Solaris_VMRaid5Component
- Solaris_VMSoftPartComponent
- Solaris_VMStatistics
- Solaris_VMStripeComponent
- Solaris_VMTransLog
- Solaris_VMTransMaster
- Solaris_VMUsesHotSparePool
- Solaris_VMVolumeBasedOn

Solaris_Project1.0.mof File

The Solaris_Project1.0.mof file defines classes that represent the Solaris project database.

The Solaris_Project1.0.mof file defines the class Solaris_Project. In addition, the Solaris_Project1.0.mof file defines the following association classes:

- Solaris_ProjectGroup
- Solaris_ProjectUser

Solaris_Performance1.0.mof File

The Solaris_Performance1.0.mof file defines classes that pertain to computing resource metrics, that is, classes that pertain to the use and performance of computing resources for each user and for each project.

The Solaris_Performance1.0.mof file defines the following classes:

- Solaris_ActiveProject
- Solaris_ActiveUser
- Solaris_ProcessStatisticalInformation
- Solaris_ProjectProcessAggregateStatisticalInformation
- Solaris_UserProcessAggregateStatisticalInformation

In addition, the Solaris_Performance1.0.mof file defines the following association classes:

- Solaris_ActiveProjectProcessAggregateStatistics
- Solaris_ActiveUserProcessAggregateStatistics
- Solaris_ProcessStatistics
- Solaris_ProjectProcessStatistics
- Solaris_UserProcessStatistics

Index

A

access control
  setting
    on a namespace, 70
    on a user, 69
application programming interfaces (APIs)
  calling methods, 63
  creating a namespace, 65
  creating instances, 49
  deleting a class, 67
  deleting instances, 50
  enumerating namespaces, 57
  overview, 20
  programming tasks, 46
  retrieving classes, 63
  setting CIM qualifiers, 71
  setting instances, 52

B

base class
  creating, 66

C

CIM Object Manager
  error messages, 121
  Repository, 139
CIM qualifiers
  setting, 71
CIM Schema
  files, 137
CIM WorkShop
  starting, 22
class
  `deleteClass`, 66
  deleting, 67
  retrieving, 63
  security, 68
classes
  `CIMClass`, 66
  creating, 66
  deleting, 67

D

default namespace, 46
Distributed Management Task Force, 17

E

Error messages, 121
examples
  calling a method, 63
  creating a namespace, 65
  creating an instance, 49
  deleting a class, 67
  deleting an instance, 50
  enumerating namespaces, 57
  retrieving a class, 63
  setting CIM qualifiers, 71

examples *(continued)*
setting instances, 52
exceptions, *See* error messages

I

instance
creating, 49
deleting, 49
getting and setting, 51

J

Java
creating instances, 49
deleting instances, 50
integrating Java programs with native methods, 103
setting instances, 52
Solaris WBEM SDK example programs, 115

M

Managed Object Format
creating base classes, 66
description, 19
Managed Object Format (MOF)
Solaris Schema, 138
method
`deleteInstance`, 49
deleting a namespace, 66
`enumNameSpace`, 57
`getClass`, 63
`getInstance`, 51
`invokeMethod`, 62
method, `createInstance`, 68
Methods
calling, 63
MOF file
security caution for compiling, 31
`mofcomp` command
security caution, 31

N

namespace
creating, 65
default, 47
enumerating, 57
setting access control, 68

P

provider
interfaces, 92
registering with the CIM Object Manager, 104
writing a native provider, 102

Q

qualifier
definition, 71
example type declaration, 71
key, 66

S

schema
CIM
files, 137
Solaris
files, 137
Solaris Schema, 138
security namespace, 46
single provider, 91
Solaris Schema
files, 137
Solaris WBEM SDK
programming tasks, 46
Solaris WBEM SDK error messages, *See* error messages
Solaris WBEM SDK example programs, *See* example programs

Solaris WBEM Services error messages, *See* error messages

W

WBEM
definition, 17
workshop
See CIM WorkShop