



Solaris WBEM SDK Developer's Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-6828
December 2001

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. Tous droits réservés

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



011025@2471



Contents

Preface	15
1 Overview of Web-Based Enterprise Management	21
About WBEM	21
Sun WBEM SDK	22
About CIM	22
CIM Terms and Concepts	22
CIM Structure	25
How it All Fits Together	31
About Managed Object Format	32
MOF Syntax	32
Schema MOF Files	33
2 CIM WorkShop	35
About CIM WorkShop	35
Reference: CIM WorkShop Interface	36
Login Dialog Box	36
The CIM WorkShop Main Window	38
CIM WorkShop Menus	40
New Class Dialog Box	42
Add Property Dialog Box	43
Qualifiers Dialog Box	43
Scope and Flavors Dialog Boxes	44
Set Value Dialog Box	44
Instance Editor Window	48

Add Instance Dialog Box	51
Invoke Method Dialog Box	51
Icons on the CIM WorkShop Tool Bar	53
The Properties Tab	53
The Methods Tab	54
The Events Tab	54
Starting CIM WorkShop	55
▼ To Start CIM WorkShop	55
Navigating in CIM WorkShop	56
Browsing the Class Inheritance Tree	56
Finding a Class	57
Executing Queries	57
▼ To Execute a Query	57
Viewing Class Characteristics	58
Selecting a Class	58
Viewing Class Properties	58
Viewing Class Methods	58
Viewing Qualifiers	58
Viewing the Scope of a Qualifier	59
Viewing the Flavor of a Qualifier	59
Viewing Values	59
Working in Namespaces	59
Moving to Another Namespace	60
Creating and Deleting a Namespace	60
Moving to Another Host	61
Refreshing Classes and Namespaces	62
Adding a New Qualifier Type	62
Working with Classes	63
Adding a Class	64
Creating a Class	64
Adding Qualifiers	65
Adding New Properties to a Class	66
Adding Qualifiers	66
Adding Qualifiers to a Method	67
Deleting Classes and Their Attributes	68
Deleting a Class	68
Deleting a Property of a Class	68

Deleting Qualifiers	69
Working with Instances	70
Displaying Instances	70
Adding Instances	71
Deleting Instances	72
Invoking Methods	73
▼ To Invoke a Method	73
3 Application Programming Interfaces	75
About the APIs	75
API Packages	76
4 Writing Client Applications	79
Overview	79
Sequence of a Client Application	80
Sample Solaris WBEM SDK Client Application	80
Typical Programming Tasks	82
Opening and Closing a Client Connection	82
About Namespaces	82
Connecting to the CIM Object Manager	83
Closing a Client Connection	85
Working with Instances	85
Creating an Instance	85
Deleting an Instance	86
Getting and Setting Instances	88
Enumerating Namespaces, Classes, and Instances	92
Deep and Shallow Enumeration	93
Getting Class and Instance Data	93
Getting Class and Instance Names	94
Enumerating Namespaces	94
Enumerating Class Names	96
Querying	98
The execQuery Method	99
Using the WBEM Query Language	99
Making a Data Query	101
About Associations	103

The Association Methods	103
Working With the <code>associators</code> and <code>associatorNames</code> Methods	107
Working With the <code>references</code> and <code>referenceNames</code> Methods	108
Calling Methods	109
Calling a Method	109
Retrieving Class Definitions	110
Handling Exceptions	111
Using the Try/Catch Clauses	111
Syntactic and Semantic Error Checking	112
Advanced Programming Topics	112
Creating a Namespace	112
Deleting a Namespace	113
Creating a Base Class	115
Deleting a Class	117
Working with Qualifier Types and Qualifiers	119
Sample Programs	120
5 Writing a Provider Program	123
About Providers	123
Types of Providers	124
Implementing the Provider Interfaces	124
Implementing an Instance Provider	125
Implementing a Property Provider	127
Implementing a Method Provider	128
Implementing an Association Provider	129
Writing a Native Provider	130
Installing a Provider	131
▼ How To Set Up the Environment	131
Setting the Solaris Provider <code>CLASSPATH</code>	131
Registering a Provider	132
▼ How To Register a Provider	133
Changing a MOF File	133
Registering a Provider	134
Modifying a Provider	135
▼ How To Modify a Provider	135
Handling WBEM Query Language Queries	135
Using the WQL APIs to Parse Query Strings	136

Writing a Provider that Parses WQL Query Strings 139

6	Handling CIM Events	143
	The CIM Event Model	143
	How Indications are Generated	144
	How Subscriptions are Created	146
	Creating a Subscription	146
	Adding a CIM Listener	147
	Creating an Event Filter	147
	Creating an Event Handler	149
	Binding an Event Filter to an Event Handler	150
	Generating an Event Indication	151
	Methods in the EventProvider Interface	151
	Creating and Delivering Indications	152
	Authorizations	153
	CIM Indication Classes	153
7	Using the Solaris WBEM SDK Sample Programs	155
	About the Sample Programs	155
	Running the Sample Applet	156
	▼ How To Run the Sample Applet Using Appletviewer	156
	▼ How To Run the Sample Applet using a Web Browser	156
	About the Client Sample Programs	156
	Running the Client Sample Programs	158
	About the Provider Sample Program	159
	Running the Provider Sample Program	160
A	WBEM Error Messages	163
	About WBEM Error Messages	163
	Parts of an Error Message	163
	Error Message Templates	164
	List of Error Messages	164

Glossary 181

Index 187

Tables

TABLE 1-1	Core Model Logical Elements	26
TABLE 1-2	Core Model System Classes	27
TABLE 2-1	The CIM Workshop Main Window	39
TABLE 2-2	CIM WorkShop Menus and Menu Selections	40
TABLE 2-3	Fields on the Qualifiers Dialog Box	43
TABLE 2-4	Qualifiers Dialog Box Buttons	44
TABLE 2-5	Icons on the Instance Editor Window Tool Bar	49
TABLE 2-6	Menus on the Instance Editor Window	50
TABLE 2-7	Icons on the CIM WorkShop Tool Bar	53
TABLE 4-1	Deep and Shallow Enumeration	93
TABLE 4-2	Mapping of SQL to WQL Data	100
TABLE 4-3	Supported WQL Key Words	100
TABLE 4-4	WQL Operators	101
TABLE 4-5	SELECT Statement	101
TABLE 4-6	The CIMClient Association Methods	104
TABLE 4-7	Optional Arguments to the Association Methods	106
TABLE 4-8	associators and associatorNames Methods	107
TABLE 4-9	references and referenceNames Methods	108
TABLE 4-10	Parameters to the invokeMethodMethod	109
TABLE 6-1	Properties in the CIM_IndicationFilter Class	147
TABLE 6-2	Properties in the CIM_IndicationHandler Class	149
TABLE 6-3	Methods in the EventProvider Interface	151
TABLE 6-4	CIM Events Indication Classes	153
TABLE 7-1	Client Sample Programs	157
TABLE 7-2	Provider Sample Files	159

Figures

FIGURE 2-1	CIM Workshop Login Dialog Box	36
FIGURE 2-2	The CIM WorkShop Main Window	38
FIGURE 2-3	The New Class Dialog Box	42
FIGURE 2-4	The Instance Editor Window	48
FIGURE 2-5	The Instance Editor Window Tool Bar	49
FIGURE 2-6	The Invoke Method Dialog Box	51
FIGURE 2-7	The CIM WorkShop Tool Bar	53
FIGURE 4-1	An Association Between Teacher and Student	103
FIGURE 4-2	Teacher-Student Association Example	107
FIGURE 5-1	WBEM Classes that Represent the WBEM Query Language Expression	136

Examples

EXAMPLE 4-1	Sample Solaris WBEM SDK Application	80
EXAMPLE 4-2	Connecting to the Default Namespace	84
EXAMPLE 4-3	Connecting to the Root Account	84
EXAMPLE 4-4	Connecting to a Non-Default Namespace	84
EXAMPLE 4-5	Authenticating as an RBAC Role Identity	85
EXAMPLE 4-6	Creating an Instance	86
EXAMPLE 4-7	Deleting Instances	87
EXAMPLE 4-8	Getting Instances of a Class	89
EXAMPLE 4-9	Printing Processor Information (<code>getProperty</code>)	90
EXAMPLE 4-10	Setting a Property	91
EXAMPLE 4-11	Setting Instances	92
EXAMPLE 4-12	Enumerating Namespaces	94
EXAMPLE 4-13	Enumerating Class Names	96
EXAMPLE 4-14	Enumerating Class Data	96
EXAMPLE 4-15	Enumerating Classes and Instances	96
EXAMPLE 4-16	<code>execQuery</code> Example	99
EXAMPLE 4-17	Passing Instances to the <code>Associators</code> Method	105
EXAMPLE 4-18	Calling a Method	109
EXAMPLE 4-19	Retrieving a Class Definition	110
EXAMPLE 4-20	Semantic Error Checking	112
EXAMPLE 4-21	Creating a Namespace	113
EXAMPLE 4-22	Deleting a Namespace	114
EXAMPLE 4-23	Creating a CIM Class	116
EXAMPLE 4-24	Deleting a Class	118
EXAMPLE 4-25	Getting CIM Qualifiers	119
EXAMPLE 4-26	Set Qualifiers	120

EXAMPLE 5-1	SimpleInstanceProvider Instance Provider	125
EXAMPLE 5-2	Implementing a Property Provider	127
EXAMPLE 5-3	Implementing a Method Provider	128
EXAMPLE 5-4	Implementing an Association Provider	129
EXAMPLE 5-5	SimpleInstanceProvider	134
EXAMPLE 5-6	Provider that Implements the execQuery Method	140
EXAMPLE 6-1	Adding a CIM Listener	147
EXAMPLE 6-2	Creating a CIM Event Handler	150
EXAMPLE 6-3	Binding an Event Filter to an Event Handler	150
EXAMPLE 7-1	Running the SystemInfo Client Program	158

Preface

The *Solaris WBEM SDK Developer's Guide* describes the Sun Web-Based Enterprise Management Software Developer's Toolkit, which enables software 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 intended for the following developers:

- **Instrumentation Developers**
Instrumentation developers provide resources such as processors, memory, routers, and other manageable devices. These developers communicate device information in a standard CIM format to the CIM Object Manager, typically through a software *provider*.
- **System and Network Application Developers**
System and Network Application Developers write applications that manage the information stored in CIM classes and instances. These developers typically use the Solaris WBEM Application Programming Interfaces to get and set the properties of predefined CIM instances and classes.

Before You Read This Book

This book describes how to use the Solaris WBEM SDK components and tools to write management applications.

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

The following Web sites are useful resources when working with WBEM technologies.

- Distributed Management Task Force (DMTF)
Visit www.dmtf.org for the latest developments on CIM, information about various working groups, and contact information for extending the CIM Schema.
- Rational Software
Visit www.rational.com/uml for documentation on the Unified Modeling Language (UML) and the Rose CASE tool.

How This Book Is Organized

Chapter 1 introduces Web-Based Enterprise Management (WBEM) and the Common Information Model (CIM).

Chapter 2 describes how to use CIM WorkShop to manipulate CIM classes, instances, methods, and properties.

Chapter 3 provides an overview of the client Application Programming Interfaces and gives examples of how to use the APIs to create and manipulate CIM objects.

Chapter 4 explains how to use the Client APIs to write client applications.

Chapter 5 describes a provider, provides an overview of the provider APIs, and explains how to write a provider.

Chapter 6 describes the CIM event model, how providers generate CIM events, and how applications subscribe to the notification of the occurrence of CIM events.

Chapter 7 explains how to run the code examples provided with the Solaris WBEM SDK.

Appendix A explains the error messages returned by the Solaris WBEM SDK APIs.

Glossary defines terms found in the WBEM documentation.

Related Information

You may also want to refer to the following related documentation.

- *Solaris WBEM Services Administration Guide* – Explains Common Information Model (CIM) concepts, and describes how to administer Web-based Enterprise Management (WBEM) services in the Solaris™ operating environment.
- Javadoc reference pages – Describes the WBEM Application Programming Interfaces at `/usr/sadm/lib/wbem/doc/index.html`
- CIM/Solaris Schema – Describes the CIM and Solaris Schema classes at `/usr/sadm/lib/wbem/doc/mofhtml/index.html`

Ordering Sun Documents

Fatbrain.com, the Internet's most comprehensive professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at <http://www1.fatbrain.com/documentation/sun>.

Accessing Sun Documentation Online

The docs.sun.comSM 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 <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name% su</code> Password:
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You must be <i>root</i> 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	#

Overview of Web-Based Enterprise Management

This chapter provides a description of Web-Based Enterprise Management (WBEM) and includes the following topics:

- “About WBEM” on page 21
- “About CIM” on page 22
- “How it All Fits Together” on page 31
- “About Managed Object Format” on page 32

About WBEM

Tip – This section provides a broad overview of WBEM. For more in-depth information, read about the Distributed Management Task Force’s “Web-Based Enterprise Management (WBEM) Initiative” at http://www.dmtf.org/standards/standard_wbem.php

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 develop 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.

Sun WBEM SDK

The Sun WBEM SDK contains the components required to write management applications that communicate with WBEM-enabled management devices. All management applications developed using the SUN WBEM SDK run on the Java platform. You can use the Sun WBEM SDK as a standalone application, or with Solaris WBEM Services.

The Sun WBEM API documentation is delivered in Javadocs and is installed at `/usr/sadm/lib/wbem/doc/index.html`.

About CIM

Tip – This section provides a broad overview of CIM. For a complete overview of CIM, including white papers and a tutorial, refer to the DMTF Education Overview at <http://www.dmtf.org/education/index.php>.

The Common Information Model (CIM), developed by the DMTF, is a core set of standards that make up WBEM. CIM is a standard approach to managing systems and networks, providing a common conceptual framework that classifies and defines the parts of a networked environment, and depicts how these various parts integrate. The model captures notions that are applicable to all areas of management, independent of technology implementation. CIM is comprised of the *CIM Specification* and the *CIM Schema*.

The CIM Specification defines the language and methodology for integration with other management models, and the CIM Schema provides the actual model descriptions for systems, applications, networks (LAN) and devices.

CIM Terms and Concepts

To understand CIM, you must first understand the concept of object-oriented modeling and how the various objects within the CIM schema relate to one other.

Object-oriented modeling is a method of representing something in the real world. The goal of object-oriented modeling is to set a representation of a physical entity into a framework, or model, that expresses the qualities and functions of the entity and its relationships with other entities. In the context of CIM, object-oriented modeling is used to model hardware and software elements.

CIM conventions for rendering a model are based on the diagrammatic concepts of the *Uniform Modeling Language (UML)*. The UML is a third-generation language for specifying, visualizing, and documenting the artifacts of an object-oriented system. UML uses shapes to represent physical entities, and lines to represent relationships. For example, in UML, classes are represented as rectangles. Each rectangle contains the name of the class it represents. A line between two rectangles represents a relationship between the two classes. A forked line that joins two classes to a higher-level class represents an association.

The following entities are represented:

Schema

The terms schema, model, and framework are synonymous; they are abstract representations of an entity that has a physical or logical existence. In CIM, a schema is a defined collection of classes used for naming and administration. Within a schema, classes and their subclasses are represented hierarchically using the following syntax:

```
Schemaname_classname.propertyname.
```

Each class name within a schema must be unique. For example, Sun Microsystems' WBEM includes a Solaris Schema, containing all classes specific to the Solaris extension of CIM. You can view the Solaris Schema at </usr/sadm/lib/wbem/doc/mofhtml/index.html>.

Class

Classes relate to managed objects or the associations between managed objects. If you need to instrument a managed object, you abstract the characteristics and properties of that object into a CIM class which can then be viewed as a template for actual instances of the managed object.

Classes can be abstract or concrete. Concrete classes have actual instances associated with them. In Solaris WBEM services, a class is expressed as an instantiation of the `CIMClass` Java class.

Instance

Instances of CIM classes can be static or dynamic. Static instances are stored in CIM repository, while dynamic instances are generated on demand by accessing the managed object through a provider application. In Solaris WBEM services, an instance is expressed as an instantiation of the `CIMInstance`

Property

Properties define the characteristics of a class. Properties of CIM classes are manipulated using the `CIMProperty` class in the Java CIM API. Properties have associated values and datatypes. Each property in a class has a unique name. A property in a class can be overridden by its subclass. A special type of property is the key property, which assists in distinguishing one instance of a class from another instance of the same class. Every concrete class in CIM must have at least one key property.

Method

A method is an operation, together with its signature. The signature consists of a possibly empty list of parameters and a return type. Methods define additional behavioral characteristics of a managed object and cover operational semantics that cannot be expressed as enumerations, retrieval, modification or deletions of instances of classes. There are no restrictions on a method's type of parameters, other than they must be one of the valid CIM data types. Within the context of a class, each method must have a unique name. Methods are inherited by subclasses and can be overridden by subclasses.

Qualifier

Qualifiers are meta data about a property, method, method parameter, class, or instance, but are not part of the definition itself. For example, a qualifier is used to indicate whether a property value is modifiable using the `WRITE` qualifier. Qualifiers always precede the declaration to which they apply. Certain qualifiers are well known and cannot be redefined. However, you may use arbitrary qualifiers. Qualifiers can be transmitted automatically from classes to derived classes, or from classes to instances, subject to certain rules.

The rules for how qualifier transmission occurs are attached to each qualifier and are encapsulated in the concept of the qualifier *flavor*. For example, you could designate a qualifier in the base class as automatically transmitted to all of its derived classes, or you could designate it as belonging specifically to that class.

In addition, you can use the qualifier flavor to control whether or not derived classes can override the qualifier value, or whether the value is fixed for an entire class hierarchy. This aspect of qualifier flavor is referred to as *override permissions*. You indicate the qualifier flavor using an optional clause after the qualifier preceded by a colon. Flavors consist of some combination of the key words `EnableOverride`, `DisableOverride`, `ToSubclass` and `Restricted`, indicating the applicable propagation and override rules.

Override

An override relationship indicates the substitution of a property or method inherited from a subclass for a property or method inherited from the superclass. In CIM, guidelines determine what qualifiers of properties and methods can be overridden. For example, if the qualifier type of a class is flagged as a key, then the key cannot be overridden, because CIM guidelines specify that a key property cannot be overridden.

Association

An association is a class that represents a relationship between two or more classes. Associations enable the creation of multiple relationship instances for a given class. System components can be related in many different ways, and associations provide a way of representing the relationships of these components. You can establish a relationship between classes without affecting any of the related classes. Only associations can have references.

Reference and Range

Reference and range define the role of an object involved in an association. The *reference* specifies the role name of the class in the context of the association. The domain of a reference is an association. The *range* of a reference is a character string that indicates the reference type.

CIM Structure

The Common Information Model categorizes information from general to specific, and consists of three information models:

- 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.

Note – Collectively, the Core Model and the Common Model are referred to as the CIM Schema.

- Extension Schema – Represents technology/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.

Solaris WBEM Services installs the CIM Schema and extensions at:
`/usr/sadm/lib/wbem/doc/mofhtml/index.html.`

The Core Model

Tip – For more in-depth information about the Core Model, see the Core Model White Paper at <http://www.dmtf.org/education/whitepapers.php>

The Core Model provides classes and associations that you can use to develop applications in which systems and their functions are represented as managed objects. These classes and associations embody the characteristics unique to all elements that comprise a system: *physical* and *logical* elements.

- Physical elements refer to the qualities of occupying space and conforming to the elementary laws of physics.
- Logical elements represent abstractions used to manage and coordinate aspects of the physical environment, such as system state or system capabilities. The Core Model logical elements are described in the following table.

TABLE 1-1 Core Model Logical Elements

Element	Description
Systems	Grouping of logical elements. Because systems are themselves logical elements, a system can be composed of other systems.
Network Components	Classes that provide a topological view of a network.
Services and Access Points	Provide a mechanism for organizing the structures that provide access to the capabilities of a system.

TABLE 1-1 Core Model Logical Elements (Continued)

Element	Description
Devices	Abstraction or emulation of a hardware entity, that may or may not be realized in physical hardware.

Core Model System Classes

The following table lists the classes that represent system aspects of the Core schema. The instances of these classes most often belong to the descendents of the objects contained within the class.

TABLE 1-2 Core Model System Classes

Class	Description	Example
ManagedSystemElement	Base class for the system element hierarchy. Any distinguishable component of a system is a candidate for inclusion in this class.	Software components, such as files; devices, such as disk drives and controllers, and physical components, such as chips and cards.
LogicalElement	Base class for all the components of the System that represent abstract system components	Profiles, processes, or system capabilities in the form of logical devices.
System	LogicalElement that aggregates an enumerable set of ManagedSystemElements. The aggregation operates as a functional whole. Within any particular subclass of System, there is a well-defined list of ManagedSystemElement classes, whose instances must be aggregated.	Local Area Network, Wide Area Network, subnet, intranet
Service	LogicalElement that contains the information necessary to represent and manage the functionality provided by a Device and/or SoftwareFeature. A Service is a general-purpose object used to configure and manage the implementation of functionality; it is not the functionality itself.	Printer, modem, fax machine

Core Model Association Classes

Association classes define the relationships shared by other classes. Association classes are flagged with an *ASSOCIATION* qualifier that denotes the purpose of the class. An association class must have at least two references—the names of the classes that share the particular relationship. Instances of an association always belong to the association class.

Associations can have the following types of relationships:

- One to one
- One to many
- One to zero
- Aggregation, such as a containment relationship between a system and its parts

Associations express the relationship between a system and the managed elements that make up the system. The CIM Schema defines two types of associations:

- Component associations – Indicate that one class is part of another.
- Dependency associations – Indicate that a class cannot function or exist without another class.

These association types are abstract, which means that association classes do not have instances alone. Instances must belong to one of an association class's descendent classes.

Component Associations

Component associations express the relationship between the parts of a system and the system itself. Component associations describe the elements that make up a system. Abstract classes that express component associations are used to create concrete associations of this type in descendent classes. The descendent concrete associations specify the composition relationships that the component, or class, has with other components.

In its most specialized role, the component association expresses the relationship between a system and its logical and physical parts.

Dependency Associations

Dependency associations establish the relationships between objects that rely on one another. The Core Model provides for the following types of dependencies:

- Functional – The dependent object cannot function without the object on which it depends.

- Existence – The dependent object cannot exist without the object on which it depends.

The following types of dependencies are included in the Core Model:

- **HostedService** – An association between a *Service* and the *System* on which the functionality resides. This association is one-to-many. A *System* may host many *Services*. *Services* are weak with respect to hosting *Systems*. Generally, a *Service* is hosted on the *System* where the logical devices or software features that implement the *Service* are located. The model does not represent *Services* hosted across multiple systems. This is modeled as an application system that acts as an aggregation point for *Services* that are each located on a single host.
- **HostedAccessPoint** – An association between a *Service Access Point (SAP)* and the *System* on which the SAP is provided. This association is one-to-many and is weak with respect to the *System*. Each *System* may host many SAPs. The access point of a *Service* can be located on the same or a different host from the *System* to which the *Service* provides access. This feature allows the model to depict both distributed systems (an application system with component *Services* on multiple hosts) and distributed access (a *Service* with access points hosted on other systems).
- **ServiceSAPDependency** – An association between a *Service* and a SAP which indicates that the referenced SAP is required for the *Service* to provide its functionality.
- **SAPSAPDependency** – An association between a SAP and another SAP which indicates that the latter SAP is required in order for the former SAP to utilize or connect with its *Service*.
- **ServiceAccessBySAP** – An association that identifies the access points for a *Service*. For example, a printer may be accessed by Netware, Apple Macintosh, or Microsoft Windows *Service Access Points*, potentially hosted on different *Systems*.

You can develop many extensions to the Core Model. One possible extension includes the addition of a *ManagedElement* class as an abstraction of the *ManagedSystemElement* class. You can add descendents of the *ManagedElement* class—classes that represent objects outside the managed system domain, such as *Users* or *Administrators*—to the Core Model.

The Common Model

The Common Model provides a set of base classes for technology-specific schema:

- Systems Model
- Devices Model
- Applications Management Model
- Networks Model
- Physical Model

These models are described in more detail in the following subsections.

Systems Model

The Systems Model describes the computer, application, and network systems that comprise the top-level system objects that make up the managed environment.

Devices Model

The Devices Model is a representation of the discrete logical units of the system that provide basic capabilities, such as storage, processing, communication, and input/output functions. There is a strong temptation to identify the system devices with the physical components of the system. This approach is incorrect because what is being managed is not the physical components themselves, but rather the operating system's representation of the devices.

The representation provided by the operating system does not have a one-to-one correspondence with the physical components of the system. For example, a modem might correspond to a discrete physical component, be provided by a multi-function card that supports a LAN adapter as well as a modem, or be provided by an ordinary process running on the system.

Note – It is important to understand the distinction between Logical Devices and Physical Components when using or making extensions to the model.

Applications Management Model

The Applications Management Model describes a set of details that is commonly required to manage software products and applications. This model can be used for various application structures, ranging from standalone desktop applications to a sophisticated, multiplatform, distributed, Internet-based application. Likewise, the model can be used to describe a single software product as well as a group of interdependent applications that form a business system.

A fundamental characteristic of the application model is the idea of the *application life cycle*. An application can be in one of four states:

- Deployable
- Installable
- Executable
- Executing

Networks Model

The Networks Model represents the various aspects of the network environment, including the network topology, the connectivity, and the various protocols and services that drive and provide access to the network.

Physical Model

The Physical Model represents the actual physical environment. Most of a managed environment is represented by logical objects, that is, objects that represent informational aspects of the environment rather than actual physical objects. Systems management is largely concerned with manipulating information that represents and controls the state of the system. Any impact on the actual physical environment (such as the movement of a read head on a physical drive or the starting of a fan) is likely to only happen as an indirect consequence of the manipulation of the logical environment. As such, the physical environment is typically not of direct concern.

Apart from anything else, physical parts of the system are not instrumented. Their current state (and possibly even their very existence) can only be indirectly inferred from other information about the system. In CIM, the physical model is a representation of this aspect of the environment and it is expected that it will differ dramatically from system to system and over time as technology evolves. It is also expected that the physical environment will always remain difficult to track and instrument, spawning the opportunity for a separate specialty, that of deploying applications, tools, and environments specifically aimed at providing information about the physical aspect of the managed environment.

CIM Extensions

Extension schemas are built into CIM to connect specific technologies into the model. By extending CIM, a specific operating environment such as Solaris can be made available to a large number of users and administrators. Extension schemas provide classes for software developers to build applications that manage and administer the extended technology.

How it All Fits Together

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.

A WBEM client application contacts the CIM Object Manager to establish a connection when it needs 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.

About Managed Object Format

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

You use the Managed Object Format (MOF) language to specify schema in CIM and WBEM. You define classes and instances using either Unicode or UTF-8, and place them in a file that you submit to the MOF compiler, `MofComp.exe`. The MOF compiler parses the file and adds the classes and instances defined in the file to the CIM Object Manager repository.

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

MOF Syntax

You use the CIM API to represent CIM objects, developed in MOF, as Java classes. The CIM Object Manager checks and enforces that these CIM objects comply with the current CIM Specification. In some cases, you can represent something syntactically correct in a MOF file that does not adhere to the CIM specification. The CIM Object Manager returns an error message when the MOF file is compiled.

For example, if you specify scope in the qualifier definition in a MOF file, the CIM Object Manager returns a compilation error because scope can only be specified in the definition of a CIM Qualifier Type. A CIM Qualifier cannot change the scope specified in the CIM Qualifier Type.

Schema MOF Files

When you install Solaris WBEM Services, MOF files that form the CIM Schema and the Solaris Schema populate the `/usr/sadm/mof` directory. These files are automatically compiled and run when the CIM Object Manager starts. The CIM Schema files, denoted by CIM in the file name, 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 use the `Solaris_` prefix in the file name, but otherwise follow the same file name conventions as the CIM Schema MOF files.

CIM WorkShop

This chapter explains how to use CIM WorkShop to add new properties, methods, and qualifiers to the classes and instances that you create. This chapter also describes how to set the scope and flavor of new qualifiers for new classes and instances. Topics include:

- “About CIM WorkShop” on page 35
- “Reference: CIM WorkShop Interface” on page 36
- “Starting CIM WorkShop” on page 55
- “Navigating in CIM WorkShop” on page 56
- “Executing Queries” on page 57
- “Viewing Class Characteristics” on page 58
- “Working in Namespaces” on page 59
- “Working with Classes” on page 63
- “Adding a Class” on page 64
- “Deleting Classes and Their Attributes” on page 68
- “Working with Instances” on page 70
- “Invoking Methods” on page 73

About CIM WorkShop

CIM WorkShop provides a graphical user interface through which you can view and create classes and instances. In CIM WorkShop, you can:

- View and select namespaces
- Add namespaces
- View and create classes
- Add properties, qualifiers, and methods to new classes

- View and create instances
- View and modify instance values
- Traverse associations
- Subscribe to and display information about events for a selected class
- Search for and display WBEM information
- Execute methods

CIM WorkShop is included in the Solaris WBEM SDK.

Note – Common Information Model (CIM) guidelines prevent you from modifying or editing the properties, methods, or qualifiers of CIM Schema or Solaris Schema classes. However, you can create new classes and instances of classes.

When you create a new class or instance, you can add or delete properties and qualifiers. You can also change the values of new qualifiers that you create for a new class, instance, property, or method. You cannot, however, change the values of inherited properties, methods, or qualifiers. When you add a new qualifier type, you can also add scope and flavor.

Reference: CIM WorkShop Interface

This section describes the Login dialog box and selected menus, dialog boxes, tool bars, panes, and fields of CIM Workshop.

Login Dialog Box

The CIM Workshop Login dialog box is displayed when you start CIM WorkShop.

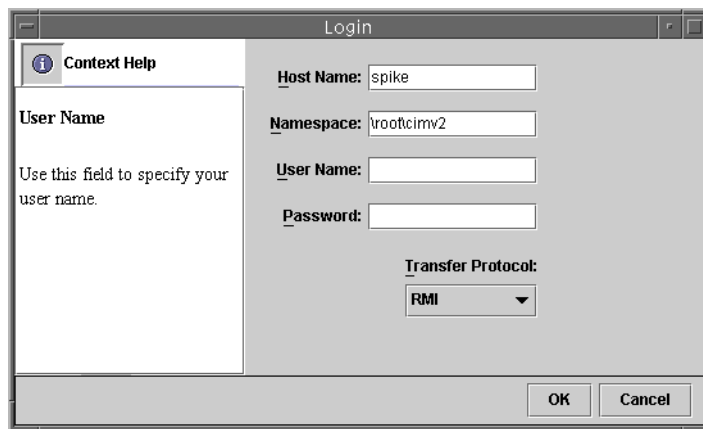


FIGURE 2-1 CIM Workshop Login Dialog Box

In the left pane on this dialog box, Context Help is displayed. When you click in a field, new information that describes how to use that particular field is displayed in the Context Help pane.

In the CIM Workshop Login dialog box, you specify:

- The host on which the CIM Object Manager is running and that contains the namespace in which you want to work
- Namespace
- Your user name
- Your password
- The transfer protocol (RMI or HTTP) that you want to use

By default, CIM WorkShop uses the remote method invocation (RMI) protocol to connect to the CIM Object Manager on the local host, in the default namespace, `root\cimv2`. You can select HTTP if you want to communicate to a CIM Object Manager by using the standard XML/HTTP protocol defined by the Desktop Management Task Force (DMTF). When your host establishes a connection, all classes that are contained in the specified namespace are displayed in the left pane on the CIM WorkShop main window.

Note – RMI is the only protocol that is supported for use with the `com.sun` application programming interface to develop WBEM software. Sun Microsystems does not support other protocols, such as XML/HTTP, for use with the `com.sun` application programming interface.

The CIM WorkShop Main Window

The CIM WorkShop main window is displayed after you log into CIM Workshop.

The CIM Workshop main window is divided into two panes. In the left pane, you can view the class inheritance tree of the current namespace. In the right pane, you can:

- View the properties and methods of a selected class.
- Specify an event for a selected class about which you want to be notified when process indication event or an instance of that event occurs.

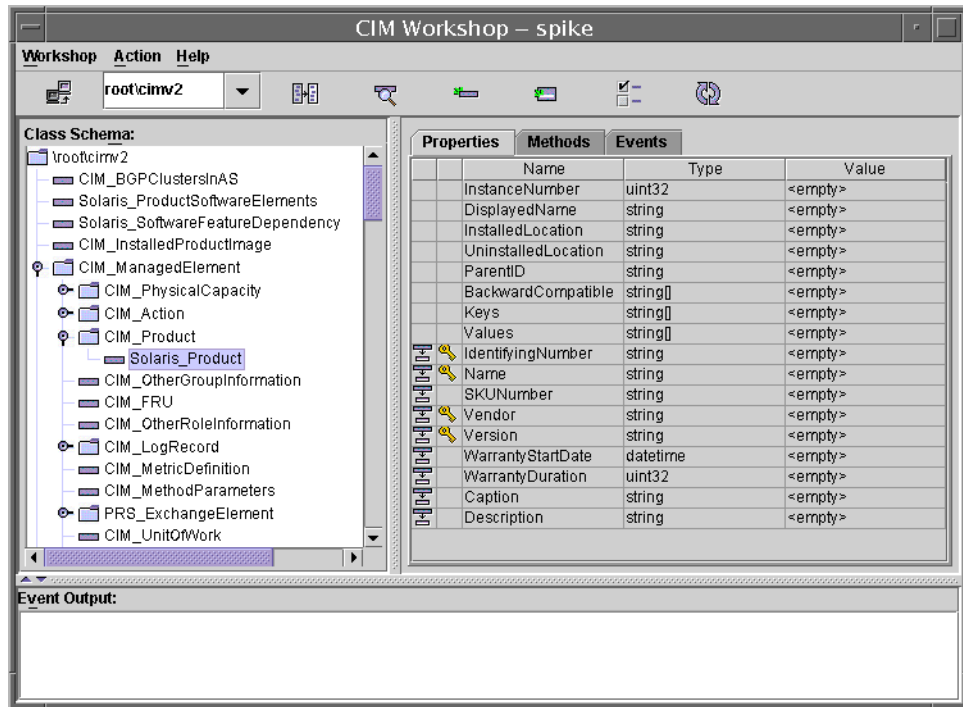
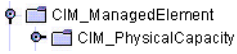


FIGURE 2-2 The CIM WorkShop Main Window

TABLE 2-1 The CIM Workshop Main Window

Element	Description
Title bar	Shows the title of the CIM WorkShop main window and the name of the host.
Menu bar	Provides a set of menus from which you can select commands that enable you to perform tasks, actions, and operations. The Menu bar also provides a menu that enables you to display information.
Tool bar	Provides icons that enable you to change hosts, change the location to a namespace that you want, find a class in the class inheritance tree, create a subclass, show instances and qualifiers of a selected class, and refresh a selected class.

TABLE 2-1 The CIM Workshop Main Window (Continued)

Element	Description
Left pane (Class Schema)	<p>Displays classes that are contained in the namespace of the current host. The left pane in the CIM WorkShop shows the contents of the selected namespace. The classes that belong to the namespace are displayed hierarchically. This organization of classes is known as a class inheritance tree.</p> <p>In the left pane, each class that contains subclasses is denoted by two icons: a folder icon and an “enabler” icon. The enabler icon looks like a knob. The enabler icon is displayed to the left of the folder icon. A folder represents a class that contains one or more subclasses. The enabler icon is a navigation aid.</p> 
Right pane	<p>Provides a Properties tab and a Methods tab, from which you can view the values of properties and methods of a class. You can view attributes and values of qualifiers and flavors by right clicking on a property or method. You can also select the options on the Events tab to subscribe to or unsubscribe from an event for a selected class.</p>
Event Output pane	<p>Displays notification when events for a class that you select occur.</p>

CIM WorkShop Menus

The following table describes the menus and menu selections on CIM WorkShop menus.

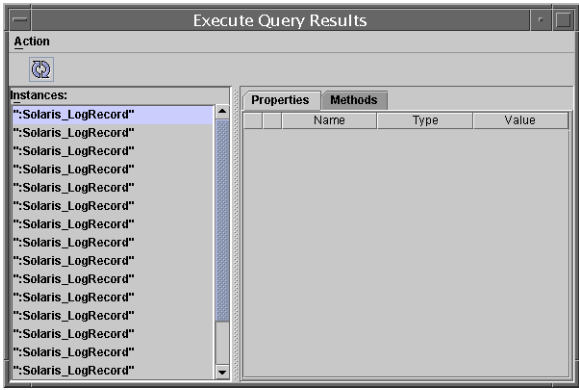
TABLE 2-2 CIM WorkShop Menus and Menu Selections

Menu	Menu Selection	Description
Workshop	Change Host	Displays the Login Dialog box, in which you specify another host and namespace that you want to use.
	Change Namespace	Displays the Change Namespace dialog box, which you use to select another namespace to use or to create or delete a namespace.
	Exit	Exits CIM Workshop.

TABLE 2-2 CIM WorkShop Menus and Menu Selections *(Continued)*

Menu	Menu Selection	Description
Action	Add Class	Displays the New Class dialog box, through which you create a subclass of a class that you select.
	Delete Class	Deletes a class that you select.
	Find Class	Displays the Input dialog box, which enables you to specify a class in the class inheritance tree for which to search.
	Show Instances	Displays the Instance Editor Window for a class that you select. All properties for the instances are shown. The Instance Editor Window enables you to add new instances of a class, delete instances of a class, and save instances of a class.
	Qualifiers	Displays the Qualifiers dialog box, which you use to view the qualifier values, scope, and flavor of the class and of the qualifiers.
	Association Traversal	Displays the Association Traversal dialog box, which enables you to view and traverse the associations of a class.
	Execute Query	Displays the Query String dialog box, which you use to specify a Level 1 WBEM Query Language (WQL) expression that you want to use in searching for WBEM information. If one or more instances of the information for which you search exist, the instances are displayed in the Execute Query Results dialog box:

TABLE 2-2 CIM WorkShop Menus and Menu Selections (Continued)

Menu	Menu Selection	Description
		
	Refresh	Retrieves the latest changes for a selected namespace, class, or subclass from the CIM Object Manager and displays those changes.
	Clear Event Output	Removes the information that appears in the Event Output pane.
Help	About	Displays the About dialog box, which you select to display the version of, and copyright information about, the Instance Editor.

New Class Dialog Box

The New Class dialog box enables you to create a new class. You display the New Class dialog box by selecting Add Class from the Action menu on the CIM Workshop main window.

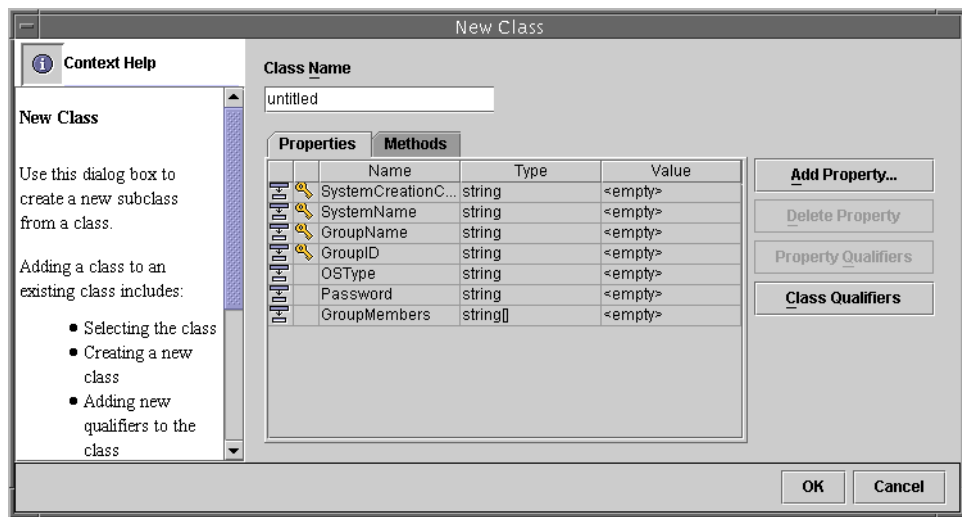


FIGURE 2-3 The New Class Dialog Box

Add Property Dialog Box

In the Add Property dialog box, you can add new properties to a class as you create it. In the Name field, you specify the name of the property. In the Type field, you select a type.

Qualifiers Dialog Box

In the Qualifiers dialog box, you can view qualifiers for a selected class, property, or method. When you create a new class, you can add qualifiers to the class or modify qualifiers of the class, its properties, or its methods in the Qualifiers dialog box. The title bar of the Qualifiers dialog box indicates the name of the class for which you view qualifiers or the class for which you add or modify qualifiers.

This table describes the fields on the Qualifiers dialog box.

TABLE 2-3 Fields on the Qualifiers Dialog Box

Name of Field	Description	Example
Name	Shows the name of the qualifier.	Provider
Type	Shows the type of value that the qualifier provides.	string

TABLE 2-3 Fields on the Qualifiers Dialog Box (Continued)

Name of Field	Description	Example
Value	Shows the value of the qualifier.	Solaris

This table describes the buttons on the Qualifiers dialog box.

TABLE 2-4 Qualifiers Dialog Box Buttons

Name of Button	Description
Scope	Displays the Scope dialog box, which you use to view the scope of a selected qualifier.
Flavors	Displays the Flavors dialog box, which you use to view the flavor of a selected qualifier.
Add Qualifier	Displays the Add Qualifier dialog box, in which you select a qualifier that you want to add for a new subclass, property, or method.
Delete Qualifier	Deletes the qualifier that you select.

Scope and Flavors Dialog Boxes

In the Scope dialog box, you view the scope of a qualifier that modifies an existing class, property, or method. In the Flavors dialog box, you view the flavor of a qualifier.

Set Value Dialog Box

When you add a new property for a class or set input parameters for a method, you use a dialog box that CIM WorkShop provides for specifying values of a particular type. The Set value dialog box is configured to accept only a value of the correct type within a particular context or null (empty value). You use the Set value dialog box to set these values:

- Unsigned Integer
- Signed Integer
- String
- Boolean
- Real Number
- DateTime
- Character
- Array
- Reference

Unsigned Integer

The Value field on this Set value dialog box accepts only an unsigned integer of a specified size or null (empty value). An unsigned integer can be a positive whole number only. CIM properties that have values that are unsigned integers can be 8 bits, 16 bits, 32 bits, or 64 bits in size. Depending on the size of the unsigned integer that makes up the value of the property, type these values in the Value field:

- For a property that is an 8-bit unsigned integer, type a positive numeric value equivalent to 8 bits.
- For a property that is a 16-bit unsigned integer, type a positive numeric value equivalent to 16 bits.
- For a property that is a 32-bit unsigned integer, type a positive numeric value equivalent to 32 bits.
- For a property that is a 64-bit unsigned integer, type a positive numeric value equivalent to 64 bits.

Signed Integer

The Value field on this Set value dialog box accepts only a signed integer of a specified size or null (empty value). A signed integer can be a negative or positive whole number. CIM properties that contain values that are signed integers can be 8 bits, 16 bits, 32 bits, or 64 bits in size. Depending on the size of the signed integer that makes up the value of the property, type these values in the Value field:

- For a property that is an 8-bit signed integer, type a positive or negative numeric value equivalent to 8 bits.
- For a property that is a 16-bit signed integer, type a positive or negative numeric value equivalent to 16 bits.
- For a property that is a 32-bit signed integer, type a positive or negative numeric value equivalent to 32 bits.
- For a property that is a 64-bit signed integer, type a positive or negative numeric value equivalent to 64 bits.

String

The Value field on this Set value dialog box accepts alphanumeric characters or null (empty value). When you specify the value of a property that is a character string, you must enter a character string, such as `Processor_Type`, in the Value field.

Boolean

You select True, False, or No Value as the value of a selected property on this Set value dialog box.

Real Number

The Value field on this Set value dialog box accepts only a real number or null (empty value). A real number can be a negative or positive number, including a decimal point. When you create a property that is of type Real Number, type a real number in the Value field.

DateTime

You set the date and the time (as specified in the CIM Specification) for a new property or method on this Set value dialog box. This Set value dialog box is displayed when you select the:

- Value of a new property that uses the DateTime value on the New Class dialog box.
- Input Value of a method's parameter on the Invoke Method dialog box.

On this Set value dialog box, you specify no value, a date, or an interval. If you specify a date, you specify these fields:

- Year *yyyy* is a 4-digit year.
- Month *mm* is the month.
- Day *dd* is the day.
- Hour *hh* is the hour, on a 24-hour clock.
- Minutes *mm* is the minute.
- Seconds *ss* is the second.
- Microseconds *mmmmmmmm* is the number of microseconds.
- Universal Coordinate Time *s* is a positive (+) or negative (-) sign that indicates the Universal Coordinated Time, or a (:).
- *utc* is the offset from UTC in minutes.

If you want to specify an interval, specify these fields:

- Days is number days, which can range from 0 through 99999999 days.
- Hour is the hour, on a 24-hour clock.
- Minutes is the minute.
- Seconds is the second.

- Microseconds is the number of microseconds, which can range from 0 through 999999.

Character

The Value field on this Set value dialog box accepts only a single, 16-bit, Universal Character Set-2 (UCS-2) character as a value or null (empty value) for a property.

Array

On this Set value dialog box, you specify an array as a value for a property. Once you specify the value you want, you can modify or delete that value. Types of components of an array that you can specify include:

- 8-Bit Unsigned Integer Array – returns a collection of positive integers equivalent to 8 bits in size.
- 16-Bit Unsigned Integer Array – returns a collection of positive integers equivalent to 16 bits in size.
- 32-Bit Unsigned Integer Array – returns a collection of positive integers equivalent to 32 bits in size.
- 64-Bit Unsigned Integer Array – returns a collection of positive integers equivalent to 64 bits in size.
- 8-Bit Signed Integer Array – returns a collection of positive or negative integers equivalent to 8 bits in size.
- 16-Bit Signed Integer Array – returns a collection of positive or negative integers equivalent to 16 bits in size.
- 32-Bit Signed Integer Array – returns a collection of positive or negative integers equivalent to 32 bits in size.
- 64-Bit Signed Integer Array – returns a collection of positive or negative integers equivalent to 64 bits in size.
- String Array – returns a collection of character strings.
- Boolean Array – returns a collection of Boolean expressions, true or false.
- 32-Bit Real Array – returns a collection of positive or negative real numbers, with or without a decimal point, equivalent to 32 bits in size.
- 64-Bit Real Array – returns a collection of positive or negative real numbers, with or without a decimal point, equivalent to 64 bits in size.
- 16-Bit Character Array – returns a collection of alphabetic and numeric character strings equivalent to 16 bits in size.
- DateTime Array – returns a collection of dates.

Reference

When you set the value of a reference to a class, you are setting an instance. On the Set value dialog box, you can choose an instance of that class or you can manually set the values of the keys.

Instance Editor Window

The Instance Editor Window lists all instances of a selected class. You can also view the properties, methods, and qualifiers that are associated with each instance.

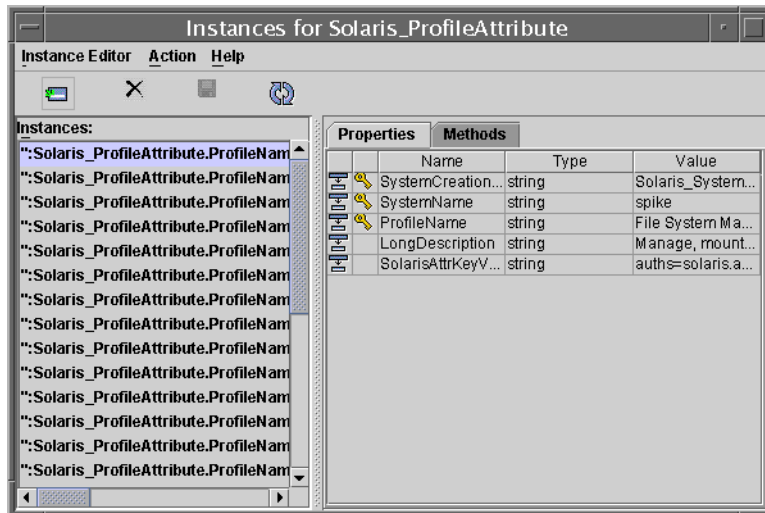


FIGURE 2-4 The Instance Editor Window

You display the Instance Editor Window by:

- Right clicking a class in the CIM WorkShop dialog box and clicking Show Instances in the pop-up menu.
- Click Action->Show Instances
- Click the Show Instances icon in the tool bar.

On the tool bar, click the Show Instances icon (the large rectangle with rays).



Panes on the Instance Editor Window

If instances of a class that you select exist, the instances are listed in the left pane on the Instance Editor Window. This dialog box contains the CIM object path (the concatenated class name and all key value pairs) of each instance. If instances do not exist, a message is displayed.

The right pane on the Instance Editor Window also contains two tabs: Properties and Methods. All properties of the selected instance are displayed in the table of the Properties tab. The Inherited Properties icon appears in the left column of the Properties table as a rectangle with an arrow pointing downwards towards another rectangle. The Inherited Properties icon indicates that the property is inherited from one of the parent classes. The Key Qualifiers icon—which appears as a key—indicates that the property has a Key qualifier.

Icons on the Instance Editor Tool Bar

The Instance Editor Window contains these icons on the tool bar:

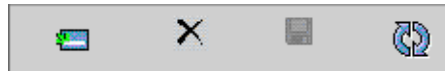


FIGURE 2-5 The Instance Editor Window Tool Bar

TABLE 2-5 Icons on the Instance Editor Window Tool Bar

Icon	Description
Add New Instance	Displays the Add Instance dialog box, which you use to add a new instance of a class.
Delete Selected Instance	Enables you to delete an instance that you select.
Save Current Instance Property Values	Updates the property values of the current instance. You can click this icon (the icon is active) only if you have made changes to instance property values.

TABLE 2-5 Icons on the Instance Editor Window Tool Bar (Continued)

Icon	Description
Refresh Instance List	Retrieves the latest changes for a class or namespace that you select from the CIM Object Manager and displays those changes.

Menus on the Instance Editor Window

This dialog box contains these menus and menu selections:

TABLE 2-6 Menus on the Instance Editor Window

Menu Name	Menu Selection	Description
Instance Editor	Exit	Closes the Instance Editor Window.
Action	Add Instance	Displays the Add Instance dialog box, which you use to add a new instance of a class.
	Delete Instance	Deletes an instance that you select.
	Save Instance	Saves an instance that you select.
	Association Traversal	Displays the Association Traversal dialog box, in which you view and traverse all the associations of an instance.
	Refresh	Retrieves the latest changes for a class or namespace that you select from the CIM Object Manager and displays those changes.
Help	About	Displays the About dialog box, which you select to display the version of, and copyright information about, the Instance Editor.

Note – You can add or delete instances only if the underlying provider enables you to add or delete instances or only if the underlying provider enables you to change property values.

Add Instance Dialog Box

In the Add Instance dialog box, if a property can be changed, you can click in the Value field to open a Set value dialog box. You cannot change the values of inherited properties.

When you add an instance, you set the key property values of that instance.

Invoke Method Dialog Box

The Invoke Method dialog box is displayed from the Methods tab of an instance of a class that contains methods. To display the Invoke Method dialog box, you right click the method and select Invoke Method from the pop-up menu that appears.

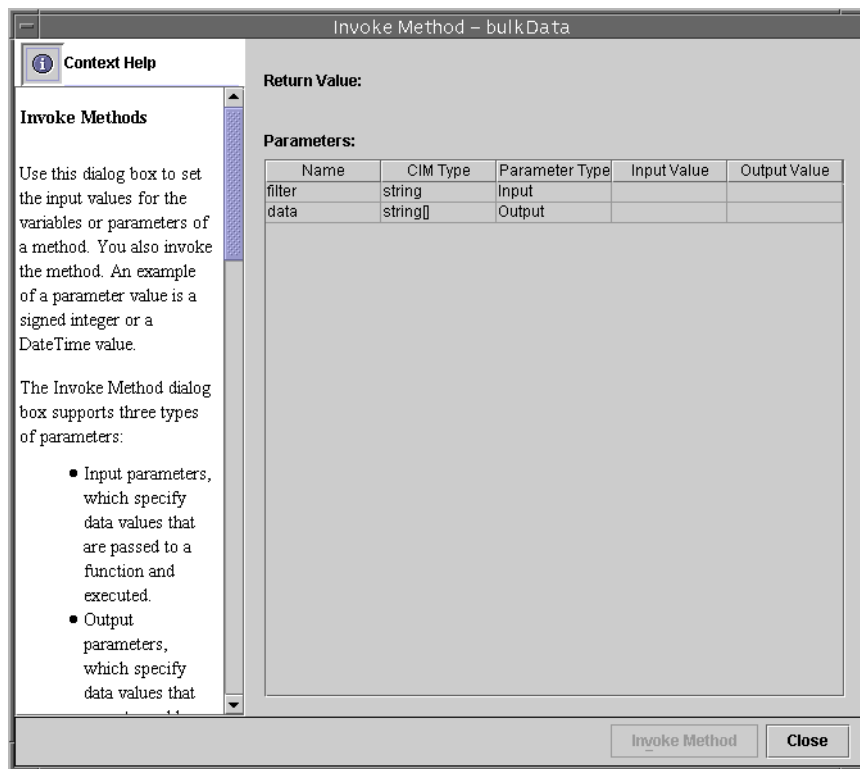


FIGURE 2-6 The Invoke Method Dialog Box

In the Invoke Method dialog box, you set the input values for the variables or parameters of a method. You also invoke the method. An example of a parameter value is a signed integer or a DateTime value.

The Invoke Method dialog box supports three types of parameters:

- Input parameters, which specify data values that are passed to a function and executed.
- Output parameters, which specify data values that are returned by a function.
- Input/Output parameters, which take data to complete functions and return values.

The Parameter Type column of the Invoke Method dialog box indicates whether the parameter of the method is input, output, or both input and output. The value of an input parameter is displayed in the Input Value column. The value of an output parameter is displayed in the Output Value column.

Information about how to invoke a method is presented in “Invoking Methods” on page 73.

Icons on the CIM WorkShop Tool Bar

The icons shown in the CIM WorkShop tool bar enable you to display and change namespaces and search for classes and instances.



FIGURE 2-7 The CIM WorkShop Tool Bar

TABLE 2-7 Icons on the CIM WorkShop Tool Bar

Icon	Description
Change Hosts	Enables you to connect to a different host or namespace and to log in with a different user name and password, and set the transfer protocol.
Select Namespace Menu Button	Enables you to select the namespace that you want to use.
Change Namespace	Displays the Change Namespace dialog box, which you use to select another namespace to use or to create or delete a namespace.
Find Class	Enables you to search for a specific class in the namespace.
Add New Class	Displays the New Class dialog box, in which you create a new subclass of a selected class.
Show Instances	Displays the Instance Editor Window. Note – You cannot show instances of an abstract class (that is, classes for which no key properties exist).
Show Qualifiers	Displays the Qualifiers dialog box, in which you view the qualifiers of a selected class.
Refresh Selected Class	Resets the display of the class hierarchy tree. Open class folders are closed and the tree is returned to the state it was in when it was first displayed.

The Properties Tab

The Properties tab in the right pane on the CIM Workshop main window provides information about a selected property. An icon that resembles a folder and an arrow pointing downwards indicates that the property is inherited from a super class. An icon that resembles a key indicates that the property is a Key property. Key properties provide unique identifiers for an instance of the class.

In the Properties tab, the Name, Type, and Value of the property are displayed. You can change the value of a property when you create a new class.

The Methods Tab

A method is a function that describes the behavior of a class. Examples of methods are behaviors such as start service, stop service, format disk, and so on. By selecting the Methods tab in the right pane, you can view all methods of the class. Methods are listed consecutively.

Methods have two parts: a signature and a body. The signature consists of the method name, the parameters names, types, and their order, and the method return type. The method body consists of a sequence of instructions.

Reading from left to right horizontally, the method contains three parts:

- Return data type — The data type of the return value for this method.
- Name of the method.
- Parameters — A comma-separated list of parameters enclosed within parentheses. Each parameter has a name and data type. Parameters that are input to the method are preceded with [IN]. Parameters that are output from the method are preceded with [OUT]. A parameter can have one or more qualifiers.

In the following example, the method `SetDateTime` takes the input parameter `Time`, which is of type `datetime` and returns a `boolean` value.

```
boolean SetDateTime([IN(true)] datetime Time);
```

The Events Tab

The Events tab in the right pane is where you select an event about which you want to be notified when an event occurs.

If the class that you select generates process indications, process indication events are displayed in the Event Output pane. You can enable or disable process indication events.

For all other classes, when you create, modify, or delete an instance of that class, a message is displayed in the Event Output pane that notes when the event occurred. You can set and unset particular events about which you want to be notified.

The actual notification that appears in Event Output pane depends on the option or options that you select on the Events tab.

Starting CIM WorkShop

You must have access to an installed CIM Object Manager to start and run CIM WorkShop. During an installation of Solaris WBEM Services in the Solaris operating environment, the CIM Object Manager runs on the local host. If you install only the Solaris WBEM SDK, you must point to a host on which the CIM Object Manager has already started. You can specify this information in the Host field of the CIM Workshop Login dialog box that is displayed when you start CIM WorkShop.

The CIM Workshop Login dialog box, the CIM WorkShop main window, and CIM Workshop menus and dialog boxes are described in “Reference: CIM WorkShop Interface” on page 36.

▼ To Start CIM WorkShop

1. At the system prompt, type:

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

The CIM Workshop Login dialog box is displayed.

2. Fill out the CIM Workshop Login dialog box:

- In the Host Name field, type the name of a host running the CIM Object Manager.

Note – By default, CIM WorkShop connects to the CIM Object Manager on the local host, in the default namespace, `root\cimv2`. If you start CIM WorkShop as part of the WBEM SDK in the Solaris operating environment or in the Microsoft Windows environment, you need to provide the name of a host that is already running a CIM Object Manager.

- In the Namespace field, click in the field and type the name of the namespace that you want to use, or retain the name of the default namespace.
- In the User Name field, type the user name you generally use for system and networking privileges.

Note – You must have write access to the current namespace to perform particular operations, such as adding or deleting a class in CIM Workshop. Setting user privileges for a particular namespace is described in more detail in “Administering Security” in *Solaris WBEM Services Administration Guide*.

- In the Password field, type the password you generally use for system and networking privileges.
- Select the transfer protocol you want to use.

3. **Click OK.**

The CIM Workshop main window is displayed. A message is displayed that states that the classes in the class inheritance tree are being enumerated.

Navigating in CIM WorkShop

When you start CIM WorkShop, the class inheritance tree is displayed in the left pane on the CIM WorkShop main window. When you select a class, its associated properties are listed in the right pane.

Information about the tool bar, menus, and layout of the CIM WorkShop main window is presented in “Reference: CIM WorkShop Interface” on page 36.

Browsing the Class Inheritance Tree

▼ To Display the Contents of a Class

- **Click the enabler icon that is displayed next to the class you want.**

▼ To Display the Properties and Methods of a Class

- **Click the folder icon that is displayed next to the class you want.**

The properties and methods of the class are displayed in the right pane on the CIM WorkShop main window.

Finding a Class

CIM WorkShop enables you to search for a specific class.

▼ To Find a Class

1. Display the Find Class dialog box:

- From the menu bar, select Action->Find Class.
- On the tool bar, click the Find Class icon (the magnifying glass).



- Press the keys Alt+a and then Alt+f.
The Find Class dialog box is displayed.

2. In the Find Class dialog box, type the name of the class for which you want to search and click OK.

If it is found, the class you specified is displayed. Details about the class for which you searched are displayed in the right pane on the CIM WorkShop main window.

Executing Queries

You need to specify a Level 1 WBEM Query Language (WQL) expression to search for WBEM information. Using WQL is described in “Querying” on page 98.

▼ To Execute a Query

1. From the CIM WorkShop main window, display the Query String dialog box:

- From the menu bar, select Action->Execute Query.
- Press the keys Alt+a and then Alt+x.

2. In the Query field, type the WQL expression that you want to use and click OK.

If one or more instances exist, the instances are displayed in the Execute Query Results dialog box.

Viewing Class Characteristics

Two tabs, Properties and Methods, which indicate the properties and methods of classes, are displayed in the right pane on the CIM Workshop main window. The Events tab, which you use to subscribe to or unsubscribe from events, is also displayed in the right pane on the CIM Workshop main window. You select a tab by clicking the tab.

Selecting a Class

You select a class by clicking the folder or page icon of the class in the class inheritance tree.

Viewing Class Properties

By default, when the CIM WorkShop main window is displayed, properties appear in the right pane on the CIM WorkShop main window. In the left pane on the CIM WorkShop main window, you can select a class in the class inheritance tree. To view all properties of that class, click the Properties tab.

Viewing Class Methods

After you select a class in the class inheritance tree, you can click the Methods tab to display the methods associated with the class.

Viewing Qualifiers

In CIM, qualifiers are attributes of classes, instances, properties, and methods. In CIM Workshop, you can view the qualifiers by right clicking a class, property, or method and clicking Qualifiers in the pop-up menu. When you click Qualifiers, the Qualifiers dialog box is displayed.

Viewing the Scope of a Qualifier

When you click the Scope button on the Qualifiers dialog box, the Scope dialog box is displayed. In the Scope dialog box, you can view the scope of a qualifier.

Viewing the Flavor of a Qualifier

When you click the Flavors button in the Qualifiers dialog box, the Flavors dialog box is displayed. In the Flavors dialog box, you can view the flavor of a qualifier.

Viewing Values

In CIM Workshop, you can view the values of classes by right clicking a class, property, or method and clicking Show value in the pop-up menu. When you click Show value, the Show value dialog box is displayed.

Working in Namespaces

A namespace is a logical entity, an abstraction of a managed object in which classes and instances can be stored. A namespace can be implemented in various forms including a directory structure, a database, or a folder.

By default, CIM WorkShop connects to the CIM Object Manager on the local host, in the default namespace `root\cimv2`. All classes that are located in the default namespace are displayed in the left pane on the CIM WorkShop main window. The name of the current namespace is listed on the tool bar in the CIM WorkShop main window. In CIM WorkShop, you can browse the classes of namespaces on different hosts and you can move to new namespaces.

When you want to set user privileges for a particular namespace, use the Solaris WBEM User Manager. The Solaris WBEM User Manager tool is described in more detail in “Administering Security” in *Solaris WBEM Services Administration Guide*.

This section here describes how to:

- Move to another namespace.
- Create a namespace.
- Delete a namespace.
- Move to another host.
- Refresh the class inheritance tree of a namespace.

Moving to Another Namespace

In the Solaris WBEM SDK, the default namespace is `root\cimv2`. You can change to another namespace if you choose.

▼ To Move to Another Namespace: Menu Selection or Icon Method

1. **In the CIM WorkShop main window, click Workshop->Change Namespace or click the Change Namespace icon on the CIM WorkShop tool bar.**

The Change Namespace dialog box is displayed.

2. **In the Namespace window on the Change Namespace dialog box, select the namespace that you want to use.**

3. **Click OK.**

The namespace that you selected becomes the current namespace.

▼ To Move to Another Namespace: Select Namespace Menu Button Method

1. **On the tool bar on the CIM WorkShop main window, click the down arrow on the Select Namespace Menu button.**

2. **On the menu, select the namespace that you want.**

The namespace that you selected becomes the current namespace.

Creating and Deleting a Namespace

You can create or delete one or more namespaces within an existing namespace.

▼ To Create a Namespace

1. **Select Change Namespace from the Workshop menu or click the Change Namespace icon in the CIM WorkShop main window.**

The Change Namespace dialog box is displayed.

2. **In the Namespace window on the Change Namespace dialog box, select the namespace in which you want to create the new namespace.**

3. Click Add.

The New Namespace dialog box appears.

4. Type the name of the new namespace that you want to create and click OK.

The new namespace that you specified is created in the namespace that you selected.

▼ To Delete a Namespace

1. Select Change Namespace from the Workshop menu or click the Change Namespace icon in the CIM WorkShop main window.

The Change Namespace dialog box is displayed.

2. In the Namespace window on the Change Namespace dialog box, select the namespace that you want to delete.

3. Click Delete.

A prompt appears that asks you to confirm that you want to delete the namespace that you selected.

4. Click OK.

The namespace that you selected is deleted.

Moving to Another Host

You can move to another host to view namespaces or processes.

▼ To Move to Another Host

1. Display the Login dialog box:

- From the menu bar, select Workshop->Change Host.
- On the tool bar, click the Change Hosts icon (the computer screens with arrow).



- Press the keys Alt+w and then Alt+c.

The Login dialog box is displayed.

2. In the **Host Name** field, type the name of the host on which the namespace in which you want to work is located.
3. Type your user name and password in the **User Name** and **Password** fields, respectively.
4. Click **OK**.

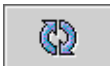
The CIM Workshop main window is displayed. The name of the host you specified is displayed in the CIM Workshop main window title bar.

Refreshing Classes and Namespaces

You can refresh the display of the class inheritance tree in the namespace to reflect current changes made by other users who work in the namespace.

▼ To Refresh a Class Inheritance Tree

1. In the class inheritance tree, click the folder of the class you want to refresh.
2. To refresh the class inheritance tree, choose one of these methods:
 - From the menu bar, select **Action->Refresh**.
 - On the tool bar, click the **Refresh Selected Class** icon (the recycle arrows).



- Press the keys **Alt+a** and then **Alt+r**.
The class inheritance tree is refreshed.

Adding a New Qualifier Type

The CIM Object Manager already contains many qualifier types. However, if you need a custom qualifier type, you can add a qualifier type to any namespace.

▼ To Add a New Qualifier Type to a Namespace

1. Move to the namespace to which you want to add a qualifier type, as described in **“Moving to Another Namespace”** on page 60.

2. In the class inheritance tree in the left pane on the CIM WorkShop main window, right click the namespace and, in the pop-up menu that appears, select Add Qualifier Type.

The Add Qualifier Type dialog box is displayed.

3. Fill out the Add Qualifier Type dialog box:

- In the Name field, type the name that you want to assign to the new qualifier type.
- In the Type selection list, select the type that you want.
- If you want to specify a default value for the new qualifier type, click Set or press the keys Alt+e, and in the Value field on the Set value dialog box that appears, type the value that you want.
- Click Scope or press the keys Alt+s, and on the Scope dialog box that appears, select the scope that you want and then click OK.
- Click Flavors or press the keys Alt+f, and on the Flavors dialog box that appears, select the flavors that you want and then click OK.

4. When you're done, on the Add Qualifier Type dialog box, click OK.

The new qualifier type is added to the active namespace.

Working with Classes

Classes are the building blocks of applications. When you start CIM WorkShop, it becomes populated with the classes that are compiled into the CIM Object Manager. These classes adhere to the DMTF standards. You cannot change their unique properties, methods, and qualifier values.

To set new values for an existing class, you can create a new instance or class. The CIM and Solaris Schema classes serve as templates. When you create a new instance or class, you produce a copy of the selected class in which you can add new properties, methods, and qualifier values. In this way, you build your own extensions into the CIM or Solaris Schema.

Note – You cannot modify the values of inherited properties, methods, or qualifiers.

Creating an instance is described in “Working with Instances” on page 70. Creating a class is described next.

Adding a Class

Adding a class to an existing class includes:

- Selecting the class
- Creating a new class
- Adding new qualifiers to the class
- Adding new properties to the class
- Adding new qualifiers to the properties
- Setting the qualifier values scope and flavor

Creating a Class

The first step in creating a class is to specify a name for the class. In CIM WorkShop, class names are displayed using standard CIM syntax: *SchemaIndicator_ClassName*. If you create a class of a CIM Schema class, the acronym *CIM* precedes the class name. If you create a class of a Solaris Schema class, the name *Solaris* precedes the class name.

Note – The underscore character (_) is required in the name of all classes.

▼ To Add a Class

1. In the class inheritance tree in the left pane on the CIM WorkShop main window, select the class from which you want to create a class.
2. Display the New Class dialog box:
 - From the menu bar, select Action->Add Class.
 - On the tool bar, click the Add New Class icon (the small rectangle with rays).



- Right click the class that you have selected in the left pane, and, on the pop-up menu that appears, click Add Class.
 - Press the keys Alt+a and then Alt+l.
- The New Class dialog box is displayed.

3. In the Class Name field, type the name of the new class.

For example, you can create a class from the class `Solaris_ComputerSystem` titled `Ultra1_ComputerSystem`.

4. Do you want to retain inherited properties and methods of the class or add a new property?

- If you want to retain inherited properties, click OK.

A class is created that uses inherited properties, methods, qualifiers, and their values.

- If you want to add a new property, click Add Property.

The Add Property dialog box is displayed.

5. Did you select Add Property in step 4?

- If no, go to the next step.

- If yes, specify the name and type of the property you want to add, and click OK.

Adding properties to a class is described in more detail in “Adding New Properties to a Class” on page 66.

6. On the New Class dialog box, click OK.

The CIM Workshop main window is displayed.

Adding Qualifiers

You can add qualifiers to a new class. You cannot change or reset the values of inherited qualifiers that modify the class. Also, you cannot delete inherited qualifiers.

▼ To Add Qualifiers

1. Display the New Class dialog box, as described in “To Add a Class” on page 64.

2. In the Class Name field, type the name of the new class that you want.

3. On the New Class dialog box, click Class Qualifiers.

The Qualifiers dialog box is displayed.

4. In the Qualifiers dialog box, right click the Qualifier for which you want to set new values and select Add Qualifier.

The Add Qualifier dialog box is displayed.

5. **In the Add Qualifier dialog box, select the name of a qualifier in the list that you want and click OK.**

The New Class dialog box is displayed.

Adding New Properties to a Class

You can add new properties to a class and modify the values of these new properties. You cannot change the values of inherited properties, and you cannot delete inherited properties.

▼ To Add a New Property to a Class

1. **Display the New Class dialog box, as described in “To Add a Class” on page 64.**

2. **In the Class Name field, type the name of the new class that you want.**

3. **On the New Class dialog box, click Add Property.**

The Add Property dialog box is displayed.

4. **In the Name field, type the name of the new property.**

5. **In the Type selection list, select the type of property that you want and click OK.**

The new property is displayed in the list under the Properties tab on the New Class dialog box.

Note – If the list of properties is long, click the scroll bar to view the property that you added.

6. **Click OK in the New Class dialog box.**

The New Class dialog box closes and the CIM Workshop main window is again displayed.

Adding Qualifiers

You can set the values of qualifiers for new properties of the class. You cannot change or reset the values of qualifiers that modify inherited properties or methods. You cannot delete inherited qualifiers.

▼ To Add Qualifiers to a New Property

1. **In the New Class dialog box, select the new property that you created in “To Add a New Property to a Class” on page 66 and click Property Qualifiers.**

The Qualifiers dialog box is displayed for the property that you created.

2. **Click Add Qualifier.**

3. **In the Name field of the Add Qualifier dialog box, select the qualifier that you want and click OK.**

The Qualifiers dialog box is displayed.

4. **On the Qualifiers dialog box, click OK.**

The New Class dialog box is displayed.

5. **On the New Class dialog box, click OK.**

The CIM Workshop main window is displayed. The qualifier and qualifier type are set for the property you selected.

Adding Qualifiers to a Method

In addition to adding qualifiers to a new property, you can also add qualifiers to a method.

▼ To Add a Qualifier to a Method

1. **From the selection list under the Methods tab on the New Class dialog box, right click the method to which you want to add a method.**

A pop-up menu is displayed.

2. **In the pop-up menu, click Qualifiers.**

The Qualifiers dialog box is displayed.

3. **Click Add Qualifier or press Alt+q.**

The Add Qualifier dialog box is displayed.

4. **Select the name of the qualifier that you want and click OK.**

The qualifier you selected is added.

5. **On the Qualifiers dialog box, click OK.**

The New Class dialog box is displayed.

Deleting Classes and Their Attributes

CIM WorkShop provides a way to delete classes, properties, methods, and qualifiers that you no longer need.

Note – When you delete a class, you delete all subclasses that it contains. You also delete all associated properties, methods, and qualifiers of the class and its subclasses.

If a class is part of an association, however, you must first delete the association before you can delete the class.

Deleting a Class

These steps describe how to delete a class from the class inheritance tree.

▼ To Delete a Class

1. **In the class inheritance tree in the left pane on the CIM Workshop main window, select the class that you want to delete.**

2. **Delete the class:**

- From the menu bar, select Action->Delete Class.
- Press the keys Alt+a and then Alt+c.

A pop-up dialog box is displayed that asks you to confirm that you want to delete the class you've selected.

3. **Click OK.**

The class that you selected is deleted.

Deleting a Property of a Class

You can delete only a property that you are creating in a new class. Otherwise, you can view, but not modify or delete, properties of classes. You cannot delete an inherited property in a subclass.

▼ To Delete a Property of a Class

1. **From the selection list under the Properties tab on the New Class dialog box, select the name of the property or the type of the property that you want to delete.**
2. **Click Delete Property or press Alt+d.**
The property is deleted.
3. **On the New Class dialog box, click OK.**
The CIM Workshop main window is displayed.

Deleting Qualifiers

When you create a new class, you can delete qualifiers of properties or methods inherited from the parent class.

▼ To Delete a Qualifier of a Property

1. **From the selection list under the Properties tab on the New Class dialog box, select the property whose qualifier you want to delete.**
2. **Click Property Qualifiers or press Alt+q.**
3. **On the Qualifiers dialog box, select the qualifier that you want to delete.**
4. **Click Delete Qualifier or press Alt+d.**
The qualifier is deleted.
5. **On the Qualifiers dialog box, click OK.**
The New Class dialog box is displayed.

▼ To Delete a Qualifier of a Method

1. **From the selection list under the Methods tab on the New Class dialog box, right click the method whose qualifier you want to delete.**
A pop-up menu is displayed.
2. **In the pop-up menu, click Qualifiers.**
The Qualifiers dialog box is displayed.
3. **Select the qualifier that you want to delete.**

4. Delete the qualifier:

- Click Delete Qualifier.
- Right click the name of the qualifier you want to delete, and, on the pop-up menu that appears, click Delete Qualifier.
- Press the keys Alt+q.

The qualifier that you selected is deleted.

5. On the Qualifiers dialog box, click OK.

The New Class dialog box is displayed.

Working with Instances

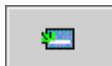
In CIM WorkShop, you can create instances of classes. You can then change the attributes of a new instance to create a unique instance of a class.

Displaying Instances

Before you create a new instance of a class, it is useful to view the instances of the class to see the properties and methods that they contain.

▼ To Display Instances of an Existing Class

1. In the class inheritance tree in the left pane on the CIM Workshop main window, select the class for which you want to view instances.
2. Display the Instance Editor Window:
 - From the menu bar, select Action->Show Instances.
 - On the tool bar, click the Show Instances icon (the large rectangle with rays).



- Right click the class that you have selected in the left pane, and, on the pop-up menu that appears, click Show Instances.
- Press the keys Alt+a+i.

The Instance Editor Window is displayed. If instances for the class you select exist, the instances are displayed in the left pane on the Instance Editor Window. All properties for the instances are shown. If instances for the class you select don't exist, the left pane on the Instance Editor Window displays nothing.

Adding Instances

▼ To Add an Instance to a Class

1. In the class inheritance tree in the left pane on the CIM Workshop main window, select the class for which you want to add instances.
2. Display the Instance Editor Window:
 - From the menu bar, select Action->Show Instances.
 - On the tool bar, click the Show Instances icon (the large rectangle with rays).

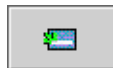


- Right click the class that you have selected in the left pane, and, on the pop-up menu that appears, select Show Instances.
- Press the keys Alt+a+i.

The Instance Editor Window is displayed. All properties for the instances are shown.

3. Add an instance:

- From the menu bar on the Instance Editor Window, select Action->Add Instance.
- On the tool bar on the Instance Editor Window, click the Add New Instance icon (the large rectangle with rays).



- In the Instances pane, right click an instance, and, on the pop-up menu that appears, select Add Instance.
- Press the keys Alt+a+d.

The Add Instance of *class* dialog box is displayed. All the keys for that class are shown.

4. **Click in the Value field for the instance key property that you want.**
The Set value dialog box is displayed.
5. **In the Value field, type the value that you want and click OK.**
The Add Instance dialog box is displayed.
6. **Repeat the previous two steps until you've set all the values in the Value column.**
7. **On the Add Instance of *class* dialog box, click OK.**
The Instance Editor Window is displayed. You've added an instance.

Deleting Instances

You can delete an instance that you don't need.

▼ To Delete an Instance

1. **In the class inheritance tree in the left pane on the CIM Workshop main window, select the class from which you want to delete an instance.**
2. **Display the Instance Editor Window:**
 - From the menu bar, select Action->Show Instances.
 - On the tool bar, click the Show Instances icon (the large rectangle with rays).



- Right click the class that you have selected in the left pane, and, on the pop-up menu that appears, select Show Instances.
 - Press the keys Alt+a+i.
The Instance Editor Window is displayed. All properties for the instances are shown.
3. **Delete the instance:**
 - From the menu bar on the Instance Editor Window, select Action->Delete Instance.
 - On the tool bar on the Instance Editor Window, click the Delete Selected Instance icon (the X).



- In the Instances pane, right click an instance, and, on the pop-up menu that appears, select Delete Instance.
- Press the keys Alt+a+s.

A pop-up dialog box is displayed that asks you to confirm that you want to delete the instance you've selected.

4. Click OK.

The instance that you selected is deleted.

Invoking Methods

In CIM WorkShop, you can set input values for a parameter of a method and invoke that method. Input parameters feed set values, such as a character string, Boolean expression, or integer to a function of the method to enable the function to complete its operations. Invoking the method returns additional data in the form of output parameters.

The dialog boxes you use to set parameter values and invoke methods are described in "Set Value Dialog Box" on page 44 and "Invoke Method Dialog Box" on page 51.

▼ To Invoke a Method

1. **In the class inheritance tree in the left pane on the CIM Workshop main window, select the class for which you want to invoke a method.**
2. **Display the Instance Editor Window:**
 - From the menu bar, select Action->Show Instances.

- On the tool bar, click the Show Instances icon (the large rectangle with rays).



- Right click the class that you have selected in the left pane, and, on the pop-up menu that appears, select Show Instances.
- Press the keys Alt+a+i.

The Instance Editor Window is displayed. Instances are displayed in the left pane on the Instance Editor Window. All properties for the instances are shown.

3. Click the Methods Tab.

The methods for the class that you selected are displayed.

4. Right click the method that you want to invoke and, on the pop-up menu that appears, select Invoke Method.

The Invoke Method dialog box is displayed.

5. Click in the field in the Input Value column for the parameter that you want to modify.

The Set value dialog box is displayed.

6. In the Value field, type the value that you want and click OK.

The Set value dialog box is dismissed. The Invoke Method dialog box is displayed. The value you specified now appears in the field that you selected earlier.

7. In each required input and output field, type the value that you want.

8. Click Invoke Method.

The return value as well as all output values are filled in automatically.

9. When you have finished adding new values and invoking the method, click Close.

The Invoke Method dialog box is dismissed. The Instance Editor Window is displayed.

10. Select Instance Editor->Exit or press the keys Alt+i+x.

The Instance Editor Window is dismissed. The CIM Workshop main window is displayed.

Application Programming Interfaces

This chapter describes the WBEM Application Programming Interfaces (APIs) and includes the following topic:

- “About the APIs” on page 75

This chapter provides a broad overview of the Sun WBEM APIs. For complete documentation on the APIs installed with the Solaris WBEM SDK, see the Javadoc reference pages at `/usr/sadm/lib/wbem/doc/index.html`.

About the APIs

The Solaris WBEM SDK applications request information or services from the Common Information Model (CIM) Object Manager through the application programming interfaces (APIs). The APIs represent and manipulate CIM objects. These APIs represent CIM objects as Java classes. An object is a computer representation or model of a managed resource, such as a printer, disk drive, or CPU. Because the CIM Object Manager enforces the Common Information Model (CIM) Specification, the objects you model using the APIs conform to standard CIM objects.

You can use these interfaces to describe managed objects and retrieve information about managed objects in a particular system environment. The advantage of modeling managed resources using CIM is that those objects can be shared across any system that is CIM-compliant.

API Packages

The following API packages are included with the Solaris WBEM SDK:

- CIM APIs – `javax.wbem.cim`
- Client APIs – `javax.wbem.client`
- Provider APIs – `javax.wbem.provider`
- Query APIs – `javax.wbem.query`

CIM API

The WBEM CIM API package, `javax.wbem.cim`, includes common classes and methods that applications use to represent all basic CIM elements. The CIM APIs create objects on the local system.

Client API

The WBEM Client API package, `javax.wbem.client`, contains classes and methods that transfer data between applications and the CIM Object Manager. Applications use the `CIMClient` class to connect to the CIM Object Manager, and use the methods to transfer data to and from the CIM Object Manager. The Client API transfers objects that have been created on the local system to the CIM Object Manager.

The Batching API represents a new addition to the Client APIs. With the addition of this API, clients can batch multiple requests in one remote call, reducing the delay introduced by multiple remote message exchanges.

Provider API

When an application requests dynamic data from the CIM Object Manager, the CIM Object Manager uses the Provider API package, `javax.wbem.provider`, to pass the request to the provider. Providers are classes that perform the following functions in response to a request from the CIM Object Manager:

- Map information from a managed device to CIM Java classes
 - Get information from a device
 - Pass the information to the CIM Object Manager in the form of CIM Java classes
- Map the information from CIM Java classes to managed device format
 - Get the required information from the CIM Java class
 - Pass the information to the device in native device format

Query API

The WBEM Query API package, `javax.wbem.query`, [need info]

Writing Client Applications

This chapter explains how to use the Client Application Programming Interfaces (APIs) to write client applications, and includes the following topics:

- “Overview” on page 79
- “Opening and Closing a Client Connection” on page 82
- “Working with Instances” on page 85
- “Enumerating Namespaces, Classes, and Instances” on page 92
- “Querying” on page 98
- “About Associations” on page 103
- “Calling Methods” on page 109
- “Retrieving Class Definitions” on page 110
- “Handling Exceptions” on page 111
- “Advanced Programming Topics” on page 112
- “Sample Programs” on page 120

For detailed information on the WBEM Client APIs (`javax.wbem.client`), see the Javadoc pages at `/usr/sadm/lib/wbem/doc/index.html`.

Overview

A Web-Based Enterprise Management (WBEM) application is a standard Java program that uses Solaris WBEM SDK APIs to manipulate CIM objects. A client application uses the CIM API to construct an object (for example, a namespace, class, or instance) and then initialize that object. The application then uses the Client APIs to pass the object to the CIM Object Manager and request a WBEM operation, such as creating a CIM namespace, class, or instance.

Sequence of a Client Application

Solaris WBEM SDK 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.
2. Use one or more APIs to perform some programming tasks.
Once a program connects to the CIM Object Manager, it uses the APIs to request operations.
3. Close the client connection to the CIM Object Manager using `CIMClient`.
Applications should close the current client session when finished. Use the `CIMClient` interface to close the current client session and free any resources used by the client session.

Sample Solaris WBEM SDK Client Application

Following is a simple client application that connects to the CIM Object Manager using all default values. The program gets a class, enumerates the instances in the class, and prints the instances.

EXAMPLE 4-1 Sample Solaris WBEM SDK Application

```
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.cim.CIMProperty;
import javax.wbem.cim.CIMValue;

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

/**
 * Returns all instances of the specified class.
 * This method takes four arguments: hostname (args[0]), username
 * (args[1]), password (args[2]) and classname (args[3]).
 */
public class WBEMsample {
    public static void main(String args[]) throws CIMException {
```


EXAMPLE 4-1 Sample Solaris WBEM SDK Application (Continued)

```
CIMClient cc = null;
/ if not four arguments, show usage and exit
if (args.length < 4) {
    System.out.println("Usage: WBEMSample host username " +
        "passwordd classname ");
    System.exit(1);
}
try {
    // args[0] contains the hostname. We create a CIMNameSpace
    // (cns) pointing to the default namespace root\cimv2 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);

    // Get the class name (args[3]) and create a CIMObjectPath
    CIMObjectPath cop = new CIMObjectPath(args[3]);

    // Do a deep enumeration of the instances of the class
    Enumeration e = cc.enumerateInstances(cop, false, true,
        true, true, null);
    // Print out all the instances of the class and its
    // subclasses.
    while (e.hasMoreElements()) {
        CIMInstance ci = (CIMInstance)e.nextElement();
        System.out.println(ci);
    } // 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();
    }
}
}
```

Typical Programming Tasks

Once a client application connects to the CIM Object Manager, it uses the API to request operations. The program's feature set determines which operations it needs to request. The tasks that most programs perform are as follows:

- Create, delete, and update instances
- Enumerate objects
- Call methods
- Retrieve class definitions
- Handle errors

In addition, applications occasionally perform the following tasks:

- Create and delete namespaces
- Create and delete classes
- Work with qualifiers

Opening and Closing a Client Connection

The first task an application performs is to establish a client session with the CIM Object Manager. WBEM Client applications request object management services from a CIM Object Manager. The client and CIM Object Manager can run on the same host or on different hosts. Multiple clients can establish connections to the same CIM Object Manager.

About Namespaces

A namespace is a directory-like structure that contains classes, instances, qualifier types, and other namespaces. The names of the objects within a namespace must be unique. All operations are performed within a namespace. When you install Solaris WBEM Services, two namespaces are created:

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

When an application connects to the CIM Object Manager, it must connect to a namespace. All subsequent operations occur within that namespace. When an application connects to a namespace, it can access the classes and instances in that namespace and in any other namespaces contained within that namespace.

For example, if you create a namespace called `child` in the `root\cimv2` namespace, you could connect to `root\cimv2` and access the classes and instances in the `root\cimv2` and `root\cimv2\child` namespaces.

An application can connect to a namespace within a namespace. This is similar to changing to a subdirectory within a directory. Once the application connects to the new namespace, all subsequent operations occur within that namespace. For example, if you open a new connection to `root\cimv2\child`, you can access any classes and instances in that namespace but cannot access the classes and instances in the parent namespace, `root\cimv2`.

Connecting to the CIM Object Manager

A client application contacts a CIM Object Manager to establish a connection each time it needs to perform a WBEM operation, such as creating a CIM class or updating a CIM instance. The application uses the `CIMClient` class to create an instance of the client on the CIM Object Manager. The `CIMClient` class takes four optional arguments:

- *namespace* – `CIMNameSpace` object that contains the names of the host name and namespace for the client connection. The default is `root\cimv2` on the local host.
- *user name* – 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 to CIM objects that is allowed. The default user account is `guest`. By default, the `guest` account allows users read access to all CIM objects in all namespaces.
- *password* – Password for the user account. The password must be a valid password for the user's Solaris account. The default password is `guest`.
- *protocol* – Protocol used for sending messages; either `RMI` (the default), or `HTTP`.

Once connected to the CIM Object Manager, all subsequent `CIMClient` operations occur within the specified namespace.

Examples — Connecting to the CIM Object Manager

The following examples show two ways of using the `CIMClient` class to connect to the CIM Object Manager.

In the first example, the application takes all the default values; it connects to the CIM Object Manager running on the local host (the same host the client application is running on), in the default namespace (`root\cimv2`), using the default user account and password, `guest`.

EXAMPLE 4-2 Connecting to the Default Namespace

```
/* Connect to root\cimv2 namespace on the local
host as user guest with password guest. */

cc = new CIMClient();
```

In this example, the application connects to the CIM Object Manager running on the local host, in the default namespace (`root\cimv2`). The application creates a `UserPrincipal` object for the root account, which has read and write access to all CIM objects in the default namespaces.

EXAMPLE 4-3 Connecting to the Root Account

```
{
    ...

    host as root. 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);
    ...
}
```

In Example 4-4, the application connects to namespace A on host `happy`. The application first creates an instance of a namespace to contain the string name of the namespace (A). Next the application uses the `CIMClient` class to connect to the CIM Object Manager, passing it the namespace object, user name, and host name.

EXAMPLE 4-4 Connecting to a Non-Default Namespace

```
{
    ...
    /* 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");
```

EXAMPLE 4-4 Connecting to a Non-Default Namespace (Continued)

```
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 4-5 Authenticating as an RBAC Role Identity

Authenticating a user's role identity requires using the `SolarisUserPrincipal` and `SolarisPasswordCredential` classes. The following examples 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

When you are finished with the client session, use the `close` method to close a session and free any resources used by the client session. The following sample code closes the client connection. The instance variable `cc` represents this client connection.

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

Working with Instances

This section describes how to create a CIM instance, delete a CIM instance, and update an instance.

Creating an Instance

Use the `newInstance` method to create an instance of an existing class. If the existing class has a key property, an application must set it to a value that is unique. 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.

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

EXAMPLE 4-6 Creating an Instance

```
...
{
/*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\cimv2namespace. */

    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. */

    ci = cimclass.newInstance();
}
...
```

Deleting an Instance

Use the `deleteInstance` method to delete an instance.

The following example connects the client application to the CIM Object Manager and uses the following interfaces to delete all instances of a class:

- `CIMObjectPath` – Constructs an object containing the CIM object path of the object to be deleted
- `enumerateInstance` – Gets the instance and all instances of its subclasses

■ deleteInstance – Deletes each instance

EXAMPLE 4-7 Deleting Instances

```
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[3])
 * 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
            CIMObjectPath cop = new CIMObjectPath(args[4]);
```

EXAMPLE 4-7 Deleting Instances (Continued)

```
// 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

An application frequently uses 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.

To create a new instance, use the `newInstance` method in the `CIMClass` class to create the instance on the local system. Then use the `CIMClient.setInstance` method to update an existing instance in a namespace or use the `CIMClient.createInstance` method to add a new instance to a namespace.

Getting Instances

The following sample code lists all processes on a given system, and uses the `enumerateInstanceNames` method to get the names of instances of the `CIM_Process` class. If you run this code on a Solaris system, the code returns Solaris processes. If you run this code on a Microsoft Windows 32 system, the code returns Windows 32 processes.

EXAMPLE 4-8 Getting Instances of a Class

```
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[3])
 * 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
```

EXAMPLE 4-8 Getting Instances of a Class (Continued)

```
// CIMNameSpace, UserPrincipal and PasswordCredential objects
// we created.
cc = new CIMClient(cns, up, pc);

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

The following sample code prints the value of the lockspeed property for all Solaris processes. This code segment uses the following methods:

- `enumerateInstanceNames` – Gets the names of all instances of the Solaris processor
- `getProperty` – Gets the value of the lockspeed for each instance
- `println` – Prints the lockspeed value

EXAMPLE 4-9 Printing Processor Information (`getProperty`)

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

EXAMPLE 4-9 Printing Processor Information (getProperty) (Continued)

```
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 lockspeed
value for each Solaris processor. */

while(e.hasMoreElements()) {
    CIMValue cv = cc.getProperty(e.nextElement(), "lockspeed");
    System.out.println(cv);
}
...
}
```

Setting a Property

The following sample code sets a hypothetical lockspeed value for all Solaris processors. This code segment uses the following methods:

- `enumerateInstanceNames` – Gets the names of all instances of the Solaris processor
- `setProperty` – Sets the value of the lockspeed for each instance

EXAMPLE 4-10 Setting a Property

```
...
{
    /* 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 and
all its subclasses. */

    Enumeration e = cc.enumerateInstanceNames(cop);

    /* Iterate through the enumeration of instance object paths.
Use the setProperty method to set the lockspeed
value to 500 for each Solaris processor. */
```

EXAMPLE 4-10 Setting a Property (Continued)

```
    for (; e.hasMoreElements(); cc.setProperty(e.nextElement(), "lockspeed",
        new CIMValue(new Integer(500))));
    ...
}
```

Setting Instances

The following sample code gets a CIM instance, updates one of its property values, and passes the updated instances to the CIM Object Manager.

A CIM property is a value used to describe a characteristic of a CIM class. Properties can be thought of as a pair of functions, one to *set* the property value and one to *get* the property value.

EXAMPLE 4-11 Setting Instances

```
...
{
    // 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);
    }
}
...
```

Enumerating Namespaces, Classes, and Instances

An enumeration is a collection of objects that can be retrieved one at a time.

The following examples show how to use the enumeration methods to enumerate namespaces, a classes, and instances.

Deep and Shallow Enumeration

The enumeration methods take a Boolean argument that can have the value *deep* or *shallow*. The behavior of *deep* and *shallow* depends upon the particular method being used, as shown in the following table.

TABLE 4-1 Deep and Shallow Enumeration

Method	<i>deep</i>	<i>shallow</i>
<code>enumClass</code>	Returns all subclasses of the enumerated class, but does not return the class itself	Returns the direct subclasses of that class
<code>enumerateInstances</code>	Returns the class instances and all instances of its subclasses	Returns the instances of that class

Getting Class and Instance Data

The following enumeration methods return the class and instance data:

- `enumerateInstances (CIMObjectPath path, boolean deep, boolean localOnly)` – Returns the instances for the class specified in *Path*. If *deep=true*, this method returns the instances of the specified class and all classes derived from the class. If *shallow=true*, this method returns the instances of the specified class.

When an instance of a class is created, it inherits the properties of the class it is derived from and all parent classes in the class hierarchy. If *localOnly=true*, `enumerateInstances` returns only non-inherited properties. If *localOnly=false*, all properties in the class are returned.

- `enumClass (CIMObjectPath path, boolean deep, boolean localOnly)` – Returns the entire class for the class specified in *Path*. If *deep=true*, this method returns all classes derived from the enumerated class. If *shallow=true*, this method returns only the first-level children of the enumerated class.

When a class is created, it inherits the methods and properties of the class it is derived from and all parent classes in the class hierarchy. If *localOnly=true*, this method returns only non-inherited properties and methods. If *localOnly=false*, all properties in the class are returned.

Getting Class and Instance Names

CIM WorkShop is an example of an application that uses enumeration methods to return the *names* of classes and instances. Once you get a list of object names, you can get the instances of an object, its properties, or other information about the object.

The following enumeration methods return the names of the enumerated class or instance:

- `enumerateInstanceNames (CIMObjectPath path)` – Returns the names of the instances for the specified class.
- `enumerateClassNames (CIMObjectPath path, boolean deep)` – Returns the names of the classes for the class specified in *Path*. If *deep=true*, this method returns the names of all classes derived from the enumerated class. If *shallow=true*, this method returns only the names of the first-level children of the enumerated class.

Enumerating Namespaces

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

EXAMPLE 4-12 Enumerating Namespaces

```
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;

/**
 *
 */
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");
        }
    }
}
```

EXAMPLE 4-12 Enumerating Namespaces *(Continued)*

```
        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 and all the namespaces it contains
    // (deep is set to true) and print each one
    Enumeration e = cc.enumNameSpace(cop, true);
    while (e.hasMoreElements()) {
        System.out.println((CIMObjectPath)e.nextElement());
    } // end while

    // Enumerate the namespace (deep = false) and print each one
    e = cc.enumNameSpace(cop, false);
    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 4-12 Enumerating Namespaces (Continued)

Enumerating Class Names

A Java GUI application might use the following code segment to display a list of classes and subclasses to a user.

EXAMPLE 4-13 Enumerating Class 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);
}
...
```

An application might use the following code segment to display the contents of a class and its subclasses.

EXAMPLE 4-14 Enumerating Class Data

```
...
{
    /* 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);
}
...
```

The following sample program performs a deep and shallow enumeration of classes and instances. This example uses the *localOnly* flag to return class and instance data instead of returning the names of the classes and instances.

EXAMPLE 4-15 Enumerating Classes and Instances

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


EXAMPLE 4-15 Enumerating Classes and Instances (Continued)

```
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]);
            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, null);
            // Will print out all the instances of the class and its
```

EXAMPLE 4-15 Enumerating Classes and Instances (Continued)

```
// 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("+++++");

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

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

Querying

The enumeration APIs return all instances in a class or class hierarchy. You can return the instance names, or the details of the instance. Querying enables you to narrow your search by specifying a query string. You can search for instances that match a

specified query in a particular class, or in all classes in 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.

The `execQuery` Method

The `execQuery` method retrieves an enumeration of CIM instances that match a query string. The query string must be formed using the WBM Query Language (WQL).

`execQuery` Syntax

The syntax for the `execQuery` method is as follows:

```
Enumeration execQuery(CIMObjectPath relNS, java.lang.String query, int ql)
```

The `execQuery` method takes the following parameters and returns an enumeration of CIM instances:

Parameter	Data Type	Description
<code>relNS</code>	<code>CIMObjectPath</code>	The namespace relative to the namespace to which you are connected. For example, if you are connected to the root namespace and want to query classes in the <code>root\cimv2</code> namespace, you pass <code>new CIMObjectPath("", "cimv2");</code>
<code>query</code>	<code>String</code>	The text of the query in WBM Query Language
<code>ql</code>	<code>String</code>	Identifies the query language, WQL

EXAMPLE 4-16 `execQuery` Example

The following `execQuery` call returns an enumeration of all instances of the `CIM_device` class in the current namespace.

```
cc.execQuery(new CIMObjectPath(), SELECT * FROM CIM_device, cc.WQL)
```

Using the WBM Query Language

The WBM Query Language is a subset of standard American National Standards Institute Structured Query Language (ANSI SQL) with semantic changes to support WBM on Solaris. Unlike SQL, WQL is currently a retrieval-only language. You cannot use WQL to modify, insert, or delete information.

SQL was written to query databases, in which data is stored in tables with a row-column structure. WQL has been adapted 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. WQL maps the CIM object model to SQL tables.

TABLE 4-2 Mapping of SQL to WQL Data

SQL	Is Represented in WQL as...
Table	CIM class
Row	CIM instance
Column	CIM property

Supported WQL Key Words

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

TABLE 4-3 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 will be used in a query.
WHERE	Narrows the scope of a query.

WBEM Query Language Operators

The following table lists the standard WQL operators that can be used in the WHERE clause of a SELECT statement.

TABLE 4-4 WQL Operators

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

Making a Data Query

Data queries are statements that request instances of classes. To issue a data query, applications use the `execQuery` method to pass a WBEM Query Language string to the CIM Object Manager.

The SELECT Statement

The SELECT statement is the SQL statement for retrieving information, with a few restrictions and extensions specific to WQL. Although the SQL SELECT statement is typically used in the database environment to retrieve particular columns from tables, the WQL SELECT statement is used to retrieve instances of a single class. WQL does not support queries across multiple classes.

The SELECT statement specifies the properties to query in an object specified in the FROM clause. The basic syntax for the SELECT statement is as follows:

```
SELECT instance FROM class
```

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

TABLE 4-5 SELECT Statement

Example Query	Description
<code>SELECT * FROM <i>class</i></code>	Selects all instances of the specified class and any of its subclasses.
<code>SELECT PropertyA FROM <i>class</i></code>	Selects only instances of the specified class and any of its subclasses that contain PropertyA.

TABLE 4-5 SELECT Statement (Continued)

Example Query	Description
<code>SELECT PropertyA, PropertyB FROM class</code>	Selects only instances of the specified class and any of its subclasses that contain PropertyA or PropertyB.

The WHERE Clause

You use the WHERE clause to narrow the scope of a query. The WHERE clause can contain a property or key word, an operator, and a constant. All WHERE clauses must specify one of the predefined WQL operators.

The basic syntax for appending the WHERE clause to the SELECT statement is as follows:

```
SELECT instance FROM class WHERE expression
```

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 property operator constant]
```

```
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"
```

The following is an invalid query:

```
SELECT * FROM PhysicalDisk WHERE Partitions < (8 + 2 - 2)
```

- 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 they have a certain amount of available space remaining, and have Solaris file systems.

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

About Associations

An association describes a relationship between two or more managed resources, for example, a computer and its hard disk. This relationship is described in an association class, which is a special type of class that contains an association qualifier. An association class also contains two or more references to the CIM instances representing its managed resources. A reference is a special property type that is declared with the `REF` keyword, indicating that it is a pointer to other instances. A reference defines the role each managed resource plays in an association.

The following figure shows two classes, `Teacher` and `Student`. Both classes are linked by the association, `TeacherStudent`. The `TeacherStudent` association has two references: `Teaches`, a property that refers to instances of the `Teacher` class and `TaughtBy`, a property that refers to instances of the `Student` class.

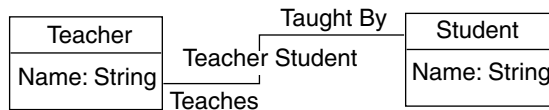


FIGURE 4-1 An Association Between Teacher and Student

You must delete an association before you delete one of its references. You can add or change the association between two or more objects without affecting the objects.

The Association Methods

The following methods in the `CIMClient` class return information about associations (relationships) between classes and instances:

TABLE 4-6 The CIMClient Association Methods

Method	Description
<code>associators</code>	Gets the CIM classes or instances that are associated with the specified CIM class or instance.
<code>associatorNames</code>	Gets the names of the CIM classes or instances that are associated with the specified CIM class or instance.
<code>references</code>	Gets the associations that refer to the specified CIM class or instance.
<code>referenceNames</code>	Gets the names of the associations that refer to the specified CIM class or instance.

Specifying the Source Class or Instance

The association methods each take one required argument, `CIMObjectPath`, which is the name of a source CIM class or CIM instance whose associations or 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 the `CIMObjectPath` is a class, the association methods return the associated classes and the subclasses of each associated class. If the `CIMObjectPath` is an instance, the methods return the associated instances and the class from which each instance is derived.

Using the Model Path to Specify an Instance

To specify the name of an instance or class, you must specify its model path. The model path for a class includes the namespace and class name. The model path for an instance uniquely identifies a particular managed resource. The model path for an instance includes the namespace, class name, and keys. A key is a property or set of properties used to uniquely identify managed resource. Key properties are marked with the `KEY` qualifier.

The model path

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

 has three parts:

- `\\myserver\\root\\cimv2` – Default CIM namespace on host `myserver`.
- `Solaris_ComputerSystem` – Name of the class from which the instances is derived.
- `Name=mycomputer, CreationClassName=Solaris_ComputerSystem` – Two key properties in the form `key property = value`.

Using the APIs to Specify an Instance

In practice, you will usually use the `enumerateInstances` method to return all instances of a given class. Then, use a loop structure to iterate through the instances. In the loop, you can pass each instance to an association method. The code segment in the following example does the following:

1. Enumerates the instances in the current class (`op`) and the subclasses of the current class.
2. Uses a While loop to cast each instance to a `CIMObjectPath` (`op`),
3. Passes each instance as the first argument to the `associators` method.

The following code example passes null or false values for all other parameters.

EXAMPLE 4-17 Passing Instances 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 to Filter Returned Classes and Instances

The association methods also take the following optional arguments, which filter the classes and instances that are returned. Each optional parameter value passes its results to the next parameter for filtering until all arguments have been processed.

You can pass values for any one or a combination of the optional arguments. You must enter a value for each parameter. The `assocClass`, `resultClass`, `role`, and `resultRole` arguments 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` arguments filter the information that is included in the classes and instances that are returned.

The following table lists the optional arguments to the association methods:

TABLE 4-7 Optional Arguments to the Association Methods

Argument	Type	Description	Value
<code>assocClass</code>	String	Returns target objects that participate in this type of association with the source CIM class or instance. If Null, does not filter returned objects by association.	Valid CIM association class name or Null
<code>resultClass</code>	String	Returns target objects that are instances of the <code>resultClass</code> or one of its subclasses, or objects that match the <code>resultClass</code> or one of its subclasses.	Valid name of a CIM class or Null
<code>role</code>	String	Specifies the role played by the source CIM class or instance in the association. Returns the target objects of associations in which the source object plays this role.	Valid property name or Null
<code>resultRole</code>	String	Returns target objects that play the specified role in the association.	Valid property name or Null
<code>includeQualifiers</code>	Boolean	If true, returns all qualifiers for each target object (qualifiers on the object and any returned properties). If false, returns no qualifiers.	True or False
<code>includeClassOrigin</code>	Boolean	If true, includes the CLASSORIGIN attribute in all appropriate elements in each returned object. If false, excludes CLASSORIGIN attributes.	True or False
<code>propertyList</code>	String	Returns objects that include only elements for properties on this list. If an empty array, no properties are included in each returned object. If NULL, all properties are included in each returned object. Invalid property names are ignored. If you specify a property list, you must specify a non-Null value for <code>resultClass</code> .	An array of valid property names. An empty array or Null

Working With the associators and associatorNames Methods

The examples in this section show how to use the `associators` and `associatorNames` methods to get information about the classes associated with the `Teacher` and `Student` classes shown in the following figure. Notice that the `associatorNames` method does 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.

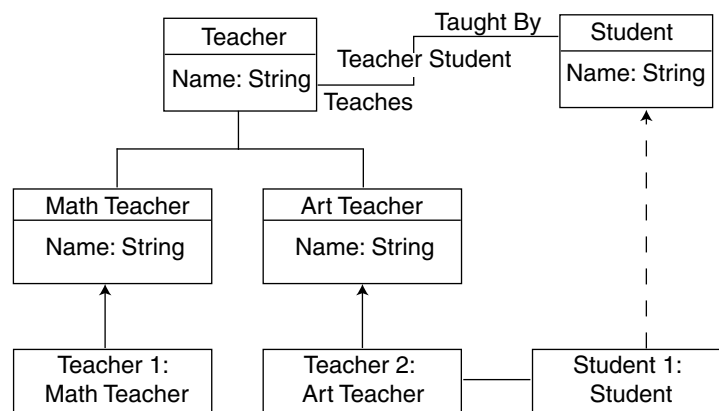


FIGURE 4-2 Teacher-Student Association Example

TABLE 4-8 associators and associatorNames Methods

Example	Output	Description
<code>associators(Teacher, null, null, null, null, false, false, null)</code>	Student class	Returns associated classes and their subclasses. <code>Student</code> is linked to <code>Teacher</code> by the <code>TeacherStudent</code> association.
<code>associators(Student, null, null, null, null, false, false, null)</code>	<code>Teacher</code> , <code>MathTeacher</code> , and <code>ArtTeacher</code> classes	Returns associated classes and their subclasses. <code>Teacher</code> is linked to <code>Student</code> by the <code>TeacherStudent</code> association. <code>MathTeacher</code> and <code>ArtTeacher</code> inherit the <code>TeacherStudent</code> association from <code>Teacher</code> .

TABLE 4-8 associators and associatorNames Methods (Continued)

Example	Output	Description
<code>associatorNames(Teacher, null, null, null, null)</code>	Name of the Student class	Returns the names of the associated classes and their subclasses. Student is linked to Teacher by the <code>TeacherStudent</code> association.
<code>associatorNames(Student, null, null, null, null)</code>	Teacher, MatchTeacher, and ArtTeacher class names.	Returns the names of the associated classes and their subclasses. Teacher is linked to Student by the <code>TeacherStudent</code> association. <code>MatchTeacher</code> and <code>ArtTeacher</code> inherit the <code>TeacherStudent</code> association from <code>Teacher</code> .

Working With the references and referenceNames Methods

The examples in this section show how to use the `references` and `referenceNames` methods to get information about the associations between the `Teacher` and `Student` classes in Figure 4-2. Notice that the `referenceNames` method does 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.

TABLE 4-9 references and referenceNames Methods

Example	Output	Comments
<code>references(Student, null, null, false, false, null)</code>	<code>TeacherStudent</code>	Returns the associations in which <code>Student</code> participates.
<code>references(Teacher, null, null, false, false, null)</code>	<code>TeacherStudent</code>	Returns the associations in which <code>Teacher</code> participates.
<code>referenceNames(Teacher, null, null)</code>	The name of the <code>TeacherStudent</code> class.	Returns the names of the associations in which <code>Teacher</code> participates.

Calling Methods

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` interface takes four arguments, as described in the following table:

TABLE 4-10 Parameters to the `invokeMethodMethod`

Parameter	Data Type	Description
<i>name</i>	<code>CIMObjectPath</code>	The name of the instance on which the method must be invoked
<i>methodName</i>	<code>String</code>	The name of the method to call
<i>inParams</i>	<code>Vector</code>	Input parameters to pass to the method
<i>outParams</i>	<code>Vector</code>	Output parameters to get from the method

The `invokeMethod` method returns a `CIMValue`. The return value is null when the method you invoke does not define a return value.

Calling a Method

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

EXAMPLE 4-18 Calling a Method

```
{
    ...
    /* 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.enumerateInstances(op, true);

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

EXAMPLE 4-18 Calling a Method (Continued)

```
Use the CIM Client getInstance class to get
the instances referred to by each object path. */

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

Retrieving Class Definitions

Use the `getClass` method to get a CIM class. When a class is created, it inherits the methods and properties of the class it is derived from and all parent classes in the class hierarchy. The `getClass` method takes the Boolean argument *localOnly*. If *localOnly* is true, this method returns only non-inherited properties and methods. If *localOnly* is false, all properties in the class are returned.

The following sample code uses these methods to retrieve a class definition:

- `CIMNameSpace` – Create a new namespace
- `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

EXAMPLE 4-19 Retrieving a Class Definition

```
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.
 */
```

EXAMPLE 4-19 Retrieving a Class Definition (Continued)

```
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 interface has a throws clause that defines a CIM Exception. An exception is an 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 API code handles.

Using the Try/Catch Clauses

The Client API uses standard Java try/catch clauses to handle exceptions. Generally, an application catches exceptions and either takes some corrective action or passes some information about the error to the user.

The CIM rules are not explicitly identified in the CIM specification. In many cases, they are implied by example. In many cases, the error code refers to a general problem, for example, a data type mismatch, but the programmer must figure out what the correct data type is for the data.

Syntactic and Semantic Error Checking

The MOF Compiler (`mofc`) compiles `.mof` text files into Java classes (bytecode). The MOF Compiler does syntactical checking of the MOF files. The CIM Object Manager does semantic and syntactical checking because it can be accessed by many different applications.

The MOF file in the following example defines two classes, A and B. If this example were compiled, the CIM Object Manager would return a semantic error because only a key can override another key.

EXAMPLE 4–20 Semantic Error Checking

```
Class A          \\Define Class A
{
  [Key]
  int a;
}
Class B:A       \\Class B extends A
{ [overrides ("c", key (false)) ]
  int b;
}
```

Advanced Programming Topics

This section describes some advanced programming operations.

Creating a Namespace

The installation compiles the standard CIM MOF files into the default namespaces, `root\cimv2` and `root\security`. If you create a new namespace, you must compile the appropriate CIM `.mof` files into the new namespace before creating objects in it. 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.

The following example uses a two-step process to create a namespace within an existing namespace.

1. The `CIMNameSpace` method constructs a namespace object that contains the parameters to be passed to the CIM Object Manager when the namespace is actually created.

2. The `CIMClient` class connects to the CIM Object Manager and passes it the namespace object. The CIM Object Manager creates the namespace, using the parameters contained in the namespace object.

EXAMPLE 4-21 Creating a Namespace

```
{
    ...
    /* 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, "root", "secret");

    /* 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). */

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

    /* Creates a new namespace called secondlevel under the
    toplevel namespace on myhost.*/

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

Deleting a Namespace

Use the `deleteNameSpace` method to delete a namespace.

The following sample program deletes the specified namespace on the specified host. The program takes five required string arguments (host name, parent namespace, child namespace, username, and password). The user running this program must specify the username and password for an account that has write permission to the namespace to be deleted.

EXAMPLE 4-22 Deleting a Namespace

```
{
import javax.wbem.cim.*;
import javax.wbem.client.CIMClient;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;

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

/**
 * This example program deletes the specified name space on the
 * specified host. The user must specify the username and
 * password of the administrative account for the CIM Object Manager
 * repository.
 */
public class DeleteNameSpace {

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

        // Initialize an instance of the CIM Client class
        CIMClient cc = null;

        // Requires a 5 command-line arguments.
        // If not all entered, prints command string.
        if (args.length < 5) {
            System.out.println("Usage: DeleteNameSpace host parentNS " +
                "childNS username password");
            System.exit(1);
        }
        try {

            /**
             * Creates a name space object (cns), which stores the host
             * name and parent name space.
             */
            CIMNameSpace cns = new CIMNameSpace(args[0], args[1]);

            /**
             * Creates the user principal and password credential used
             * to authenticate the user to the CIMOM.
             */
            UserPrincipal up = new UserPrincipal(args[3]);
            PasswordCredential pc = new PasswordCredential(args[4]);

            /**
             * Connects to the CIM Object Manager, and passes it the
             * namespace object (cns) and the user principal and password
             * credential and protocol.
             */
            cc = new CIMClient(cns, up, pc);
```

EXAMPLE 4-22 Deleting a Namespace (Continued)

```
/**
 * Creates another name space object (cop), which stores the
 * a null string for the host name and a string for the
 * child name space (from the command-line arguments).
 */

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

/**
 * Deletes the child name space under the parent name space.
 */

cc.deleteNameSpace(cop);
}
catch (Exception e) {
    System.out.println("Exception: "+e);
}

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

Creating a Base Class

Applications can create classes using either the MOF language or the client APIs. If you are familiar with MOF syntax, use a text editor to create a MOF file and then use the MOF Compiler to compile it into Java classes. This section describes how to use the client APIs to create a base class.

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. 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` (one of the predefined CIM data types).

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 MOF features as aliases, qualifiers, and qualifier flavors.

The following example creates a new CIM class in the default namespace `root\cimv2` on the local host. This class has two properties, one of which is the key property for the class. The example uses the `newInstance` method to create an instance of the new class.

EXAMPLE 4-23 Creating a CIM Class

```
{
...
    /* Connect to the root\cimv2 namespace
    on the local host and create a new class called myclass */

    // Connect to the default namespace on local host.
    CIMClient cc = new CIMClient();

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

    // Construct a new CIMClass object
    CIMClass cimclass = new CIMClass();

    // Set CIM class name to myclass.
    cimclass.setName("myclass");

    // Construct a new CIM property object
    CIMProperty cp = new CIMProperty();

    // Set property name
    cp.setName("keyprop");

    // Set property type to one of the predefined CIM data types.
    cp.setType(CIMDataType.getPredefinedType(CIMDataType.STRING));

    // Construct a new CIM Qualifier object
    CIMQualifier cq = new CIMQualifier();

    // Set the qualifier name
    cq.setName("key");

    // Add the new key qualifier to the property
    cp.addQualifier(cq);

    /* Create an integer property initialized to 10 */

    // Construct a new CIM property object
    CIMProperty mp = new CIMProperty();

    // Set property name to myprop
    mp.setName("myprop");
```

EXAMPLE 4-23 Creating a CIM Class (Continued)

```
// Set property type to one of the predefined CIM data types.
mp.setType(CIMDatatype.getPredefinedType(CIMDataType.SINT16));

// Initialize mp to a CIMValue that is a new Integer object
// with the value 10. The CIM Object Manager converts this
// CIMValue to the CIM Data Type (SINT16) specified for the
// property in the mp.setType statement in the line above.
// If the CIMValue (Integer 10) does not fall within the range
// of values allowed for the CIM Data Type of the property
// (SINT16), the CIM Object Manager throws an exception.
mp.setValue(new CIMValue(new Integer(10)));

/* Add the new properties to myclass and call
the CIM Object Manager to create the class. */

// Add the key property to class object
cimclass.addProperty(cp);

// Add the integer property to class object
cimclass.addProperty(mp);

/* Connect to the CIM Object Manager and pass the new class */
cc.createClass(new CIMObjectPath(), cimclass);

// Create a new CIM instance of myclass
ci = cc.newInstance();

// If the client connection is open, close it.
    if(cc != null) {
        cc.close();
    }
}
```

Deleting a Class

Use the `CIMClient deleteClass` method to delete a class. Deleting a class removes the class, its subclasses, and all instances of the class; it does not delete any associations that refer to the deleted class.

The following example uses the `deleteClass` method to delete a class in the default namespace `root\cimv2`. This program takes four required string arguments (host name, class name, username, and password). The user running this program must specify the username and password for an account that has write permission to the `root\cimv2namespace`.

EXAMPLE 4-24 Deleting a Class

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

Working with Qualifier Types and Qualifiers

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 Managed Object Format syntax, each CIM qualifier must have a CIM qualifier type declared in the same MOF file. Qualifiers do not have a scope attribute. Scope indicates which CIM elements can use the qualifier. Scope can only be defined in the qualifier type declaration; it cannot be changed 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 `test`. The property data type is an integer with the value `a`.

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

Getting CIM Qualifiers

The following example uses the `CIMQualifier` class to identify the CIM qualifiers in a vector of CIM elements. The example returns the property name, value, and type for each CIM Qualifier.

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 4-25 Getting CIM Qualifiers

```
{
...
} else if (tableType == QUALIFIER_TABLE) {
    CIMQualifier prop = (CIMQualifier)cimElements.elementAt(row);
    if (prop != null) {
        if (col == nameColumn) {
```

EXAMPLE 4-25 Getting CIM Qualifiers (Continued)

```
        return prop.getName();
    } else if (col == typeColumn) {
        CIMValue cv = prop.getValue();
        if (cv != null) {
            return cv.getType().toString();
        } else {
            return "NULL";
        }
    }
}
...
}
```

Setting CIM Qualifiers

The following sample code sets a list of CIM qualifiers for a new class to the qualifiers in its superclass.

EXAMPLE 4-26 Set Qualifiers

```
{
    try {
        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;
    }
}
...
}
```

Sample Programs

The examples directory contains sample programs that use the client API to perform a function. You can use these examples to start writing your own applications more quickly. The sample programs are described in Chapter 7.

To run a sample program, type the command:

java *program_name parameters*

For example, java createNameSpace *hostname username password namespaces*

Writing a Provider Program

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

- “About Providers” on page 123
- “Implementing the Provider Interfaces” on page 124
- “Installing a Provider” on page 131
- “Registering a Provider” on page 132
- “Modifying a Provider” on page 135
- “Handling WBEM Query Language Queries” on page 135

Tip – For detailed information on the WBEM Provider APIs (`javax.wbem.provider`), see the Javadoc pages at `/usr/sadm/lib/wbem/doc/index.html`.

About Providers

Providers are classes that communicate with managed resources to access data. Providers forward this information to the CIM Object Manager for integration and interpretation. When the CIM Object Manager receives a request from a management application for data that is not available from the CIM Object Manager Repository, it forwards the request to a provider.

Object providers must reside on the same machine as the CIM Object Manager. The CIM Object Manager uses provider application programming interfaces (APIs) to communicate with local providers.

When an application requests dynamic data from the CIM Object Manager, the CIM Object Manager uses the provider interfaces to pass the request to the provider.

Providers can also maintain their own data, generating it dynamically when necessary. This type of provider is known as a *pull provider*. Pull providers have minimal interaction with the CIM Object Manager. The data managed by a pull provider typically changes frequently, requiring the provider to either generate the data dynamically or retrieve it from a file or the WBEM repository whenever an application issues a request.

Providers perform the following functions in response to a request from the CIM Object Manager:

- Map the native information format to CIM Java classes
 - Get information from a device
 - Pass the information to the CIM Object Manager in the form of CIM Java classes
- Map the information from CIM Java classes to the native device format
 - Get the required information from the CIM Java class
 - Pass the information to the device in the native device format

Types of Providers

Providers are categorized according to the types of requests they service. The Solaris WBEM SDK supports the following types of providers:

- Instance – Supply dynamic instances of a given class, and support instance retrieval, enumeration, modification, and deletion.
- Property – Supply dynamic property values.
- Method – Supply methods of one or more classes.
- Association – Supply instances of dynamic association classes.
- Event – Handle indications of CIM events. For more information on event providers, see Chapter 6.

A single provider can act simultaneously as a class, instance, and method provider by proper registration and implementation of all relevant methods.

Implementing the Provider Interfaces

The provider interfaces are included in the `javax.wbem.provider` package. When the CIM Object Manager starts a provider, it calls the `initialize` method. The `initialize` method takes an argument of type `CIMOMhandle`, which is a reference

to the CIM Object Manager. The `CIMOMHandle` class contains methods that providers use to transfer data to and from the CIM Object Manager.

You can include the providers in a single Java class file or store each provider in a separate file.

Implementing an Instance Provider

The following sample code for the `SimpleInstanceProvider` instance provider implements the `enumInstances` and `getInstance` interfaces for the `Ex_SimpleInstanceProvider` class. For brevity, this example implements the `deleteInstance`, `createInstance`, `setInstance`, and `execQuery` interfaces by throwing a `CIMException`. In practice, an instance provider must implement all `InstanceProvider` interfaces.

EXAMPLE 5-1 `SimpleInstanceProvider` Instance Provider

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

public class SimpleInstanceProvider implements InstanceProvider{
    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.getObjectPath().equalsIgnoreCase(
            "Ex_SimpleInstanceProvider"))
        {

```

EXAMPLE 5-1 SimpleInstanceProvider Instance Provider (Continued)

```
        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"));
            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_SimpleInstanceProvider"))
    {
        CIMInstance ci = cc.newInstance();
        // we need to get the keys from the passed in object path, this
        // will uniquely 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);
}
```

EXAMPLE 5-1 SimpleInstanceProvider Instance Provider (Continued)

```
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)
    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));
}
}
```

Implementing a Property Provider

The following code sample creates the `fruit_prop_provider` property provider class. The `fruit_prop_provider` implements the `PropertyProvider` interface. This property provider illustrates the `getPropertyValue` method, which returns a property value for the specified parent class and property name.

Note – A CIM property is defined by its name and origin class. Two or more properties can have the same name, but the origin class uniquely identifies the property.

EXAMPLE 5-2 Implementing a Property Provider

```
...

public class SimplePropertyProvider implements PropertyProvider{
    public void initialize(CIMOMHandle cimom)
        throws CIMException {
    }

    public void cleanup()
        throws CIMException {
    }
}
```

EXAMPLE 5-2 Implementing a Property Provider (Continued)

```
public CIMValue getPropertyValue(CIMObjectPath op, string originclass,
    string propertyName){
    if (propertyName.equals("A"))
        return new CIMValue("ValueA")
    else
        return new CIMValue("ValueB");
}
...
}
```

Implementing a Method Provider

The following code sample creates a Solaris provider class that routes requests to execute methods from the CIM Object Manager to one or more specialized providers. These specialized providers service requests for dynamic data for a particular type of Solaris object. For example, the `Solaris_Package` provider services requests to execute methods in the `Solaris_Package` class. The method provider in this example implements a single method, `invokeMethod`, that calls the appropriate provider to perform one of following operations:

- Reboot a Solaris system
- Reboot or shut down a Solaris system
- Delete a Solaris serial port

EXAMPLE 5-3 Implementing a Method Provider

```
...
public class Solaris implements MethodProvider {
    public void initialize(CIMONHandle, ch) throws CIMException {
    }
    public void cleanup() throws CIMException {
    }
    public CIMValue invokeMethod(CIMObjectPath op, String methodName,
        Vector inParams, Vector outParams) throws CIMException {
        if (op.getObjectPath().equalsIgnoreCase("solaris_computersystem")) {
            Solaris_ComputerSystem sp = new Solaris_ComputerSystem();
            if (methodName.equalsIgnoreCase("reboot")) {
                return new CIMValue (sp.Reboot());
            }
        }
        if (op.getObjectPath().equalsIgnoreCase("solaris_operatingsystem")) {
            Solaris_OperatingSystem sos = new Solaris_OperatingSystem();
            if (methodName.equalsIgnoreCase("reboot")) {
                return new CIMValue (sos.Reboot());
            }
            if (methodName.equalsIgnoreCase("shutdown")) {
                return new CIMValue (sos.Shutdown());
            }
        }
    }
}
```


EXAMPLE 5-3 Implementing a Method Provider (Continued)

```
    }
    if (op.getObject().equalsIgnoreCase("solaris_serialport")) {
        Solaris_SerialPort ser = new Solaris_SerialPort();
        if (methodName.equalsIgnoreCase("disableportservice")) {
            return new CIMValue (ser.DeletePort(op));
        }
    }
    return null;
}
...
}
```

Implementing an Association Provider

A complete association provider must implement all `AssociatorProvider` methods. For brevity, the code segment in the following example implements only the `associators` method. The CIM Object Manager passes values for `assocName`, `objectName`, `role`, `resultRole`, `includeQualifiers`, `includeClassOrigin`, and `propertyList` to the association provider.

This sample code prints the name of a CIM association 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 5-4 Implementing an Association Provider

```
public Vector 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.getObject().equalsIgnoreCase(
        "example_teacher")) {
        Vector v = new Vector();
        if ((role != null) &&
            (!role.equalsIgnoreCase("teaches"))) {
            // Teachers only play the teaches role.
            return v;
        }
        // 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);
    }
}
```

EXAMPLE 5-4 Implementing an Association Provider (Continued)

```
// 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 a Native Provider

Providers get and set information on managed devices. A native provider is a machine-specific program written to run on a managed device. For example, a provider that accesses data on a Solaris system will most likely include C functions to query the Solaris 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 may not support the platform-dependent features required by your application.
- Legacy code - Often, you have Legacy code written in a programming language other than Java and want to continue to use the code with a Java provider.

The Java Native Interface (JNI) is the Java native programming interface that is part of the JDK. By writing programs using JNI, you ensure that your code is portable across all platforms. JNI enables Java code that runs within a Java Virtual Machine (JVM) 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://www.javasoft.com/docs/books/tutorial/native1.1/index.html>.

Installing a Provider

To install a provider, you must follow these steps:

1. Set up the environment. See “How To Set Up the Environment” on page 131.
2. Set the CLASSPATH. See “How To Set the Provider CLASSPATH” on page 132.
3. Become root user.
4. Stop the CIM Object Manager using `/etc/init.d/init.wbem -stop`.
5. Start the CIM Object Manager using `/etc/init.d/init.wbem -start`.
6. Exit root user.

▼ How To Set Up the Environment

1. **Set the LD_LIBRARY_PATH environment variable to the location of the provider class files and any shared library files:**

- Using the C shell:

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

Using the Bourne shell:

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

Note – If 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.

2. **Copy the shared library files to the directory specified by the LD_LIBRARY_PATH environment variable:**

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

3. **Move the provider class files to the same path as the package in which they are defined. For example, if the provider is packaged as com.sun.providers.myprovider.*:**

```
% mv *.class /usr/sadm/lib/wbem/com/sun/providers/myprovider/
```

Setting the Solaris Provider CLASSPATH

To set the Solaris provider CLASSPATH, use the client APIs to create an instance of the Solaris_ProviderPath class and set its pathurl property to the location of your

provider class files. The `Solaris_ProviderPath` class is stored in the `\root\system` namespace.

You can also set the provider `CLASSPATH` to the location of your provider class files. You can set the class path to the `jar` file or to any directory that contains the classes. Use the standard URL format that Java uses for setting the `CLASSPATH`.

Provider <code>CLASSPATH</code>	Syntax
Absolute path to directory	<code>file:///a/b/c/</code>
Relative path to directory from which the CIM Object Manager was started (/)	<code>file://a/b/c</code>

▼ How To Set the Provider `CLASSPATH`

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 your provider class files. For example:

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

3. Update the instance. For example:

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

Registering a Provider

Providers register with the CIM Object Manager to publish information about the data and operations they support, and their physical implementation. The CIM Object

Manager uses this information to load and initialize the provider and to determine the proper provider for a particular client request. All types of providers follow the same procedure for registration.

▼ How To Register a Provider

1. Create a MOF file defining a CIM class.
2. Assign the provider qualifier to the class; assign a provider name to the provider qualifier; specify the complete class name.

The provider name identifies the Java class to serve as the provider for this class. You must prepend "java" to the provider qualifier to notify the CIM Object Manager that the provider is written in the Java language. Currently, the CIM Object Manager supports only providers written in Java.

```
[Provider ("java:com.xyz.wbem.providers.provider_name")]
Class_name {
    ...
};
```

Note – Follow standard Java class and package naming conventions to create unique provider names. The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as 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, com.mycompany.wbem.myprovider.

3. Compile the MOF file:

```
% mofcomp class_name
```

For more information on using the MOF Compiler to compile a MOF file, see the *Solaris WBEM Services Administration Guide*.

Changing a MOF File

If you change a class definition in a MOF file that was previously compiled, you must delete the class from the CIM Object Manager Repository before recompiling the MOF file. Otherwise, you will get an error that the class already exists and the new

information will not propagate to the CIM Object Manager. For more information on using CIM WorkShop to delete a class, see "Deleting Classes and Their Attributes" on page 68.

Registering a Provider

The following example shows a MOF file that declares to the CIM Object Manager the `Ex_SimpleInstanceProvider` class that is served by the `SimpleInstanceProvider`. Provider and class names in a valid MOF file follow these rules:

- The class name must be a valid CIM Schema name, which means that it must have a prefix of characters, followed by an underscore, followed by more characters. For example: `green_apples` and `red_apples` are valid CIM schema names. The class names `apples` and `apples_` are not valid CIM Schema names.
- The class name must match the class name specified in the provider for the MOF file. The MOF file in the following examples declares the `Ex_SimpleInstanceProvider` class.
- The provider name specified in the MOF file must match the name of the provider class file. The MOF file in the following example specifies the `SimpleInstanceProvider` as the provider for the `Ex_SimpleInstanceProvider` class.

EXAMPLE 5-5 SimpleInstanceProvider

```
// =====  
// Title:      SimpleInstanceProvider  
// Filename:   SimpleInstanceProvider.mof  
// Description:  
// =====  
  
// =====  
// Pragmas  
// =====  
#pragma Locale ("en-US")  
  
// =====  
//   SimpleInstanceProvider  
// =====  
[Provider("java:SimpleInstanceProvider")]  
class Ex_SimpleInstanceProvider  
{  
    // Properties  
    [Key, Description("First Name of the User")]  
    string First;  
    [Description("Last Name of the User")]  
    string Last;  
};
```

Modifying a Provider

You can make changes to a provider while the CIM Object Manager and provider are running. However, you must stop and then restart the CIM Object Manager for the changes to take effect.

▼ How To Modify a Provider

1. **Edit the provider source file.**
2. **Compile the provider source file:**

```
% javac MyProvider.java
```
3. **Become root user.**
4. **Stop the CIM Object Manager.** See “Installing a Provider” on page 131.
5. **Restart the CIM Object Manager.** See “Installing a Provider” on page 131.

Handling WBEM Query Language Queries

WBEM clients use the `execQuery` method in the `CIMClient` class to search for instances that match a set of search criteria. The CIM Object Manager handles client queries for CIM data stored in the CIM Object Manager Repository and it passes to providers queries for CIM data that is served by a particular provider.

All instance providers must implement the `execQuery` interface in the `javax.wbem.provider` package to handle client queries for the dynamic data they provide. Providers can use the classes and methods in the `javax.wbem.query` package to filter WBEM Query Language (WQL) query strings. Providers with access to an entity that handles indexing can pass the query string to that entity for parsing.

Using the WQL APIs to Parse Query Strings

The classes and methods in the `javax.wbem.query` package represent a WBEM Query Language parser and the WQL string to be parsed. The package includes classes that represent clauses within the query string and methods for manipulating the strings within those clauses.

Currently, SELECT is the only type of WQL expression that you can parse. A SELECT expression contains the following parts:

- SELECT statement
- FROM clause
- WHERE clause

The WBEM Query Language Expression

The following figure and table shows the WBEM classes that represent the clauses in a WQL expression.

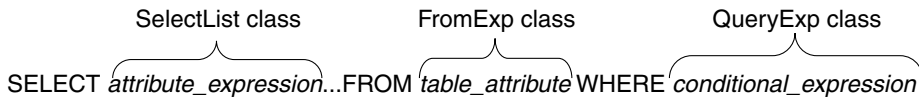


FIGURE 5-1 WBEM Classes that Represent the WBEM Query Language Expression

WBEM Query Language	WBEM Query Class
<code>SELECT attribute_expression</code>	<code>SelectList</code>
<code>FROM table_attribute</code>	<code>FromExp</code>
<code>WHERE conditional_expression</code>	<code>QueryExp</code>

WQL has been adapted 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. WQL maps the CIM object model to SQL tables, as shown in the following table:

SQL	WQL
Table	CIM class

SQL	WQL
Row	CIM instance
Column	CIM property

In CIM, a WQL expression is expressed in the following form:

SELECT *CIM property* ...**FROM** *CIM class* **WHERE** *propertyA = 40*

For example:

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

The SELECT Statement

The `SelectExp` class represents the SELECT statement. The SELECT statement is the SQL statement for retrieving information, with a few restrictions and extensions specific to WQL. Although the SQL SELECT statement is typically used in the database environment to retrieve particular columns from tables, the WQL SELECT statement is used to retrieve instances of a single class. WQL does not support queries across multiple classes.

The SELECT expression identifies the search list. The SELECT statement can take one of the following forms:

SELECT Statement	Selects
SELECT *	All instances of the specified class and any of its subclasses.
SELECT attr_exp, attr_exp...attr_exp	Only instances of the specified class and any of its subclasses that contain the specified identifiers.

The FROM Clause

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

The FROM clause identifies the class in which to search for instances that match the query string. In SQL terms, the FROM clause identifies a qualified attribute

expression, which is the name of a class to search. A qualified attribute expression identifies the table and class. Only non-joined expressions are supported, which means that a valid WQL FROM clause includes only a single class.

The WHERE Clause

The `QueryExp` class is an abstract class whose subclasses represent conditional expressions which return a boolean value when a particular `CIMInstance` is applied to them.

The WHERE clause narrows the scope of a query. The WHERE clause contains a conditional expression, which can contain a property or key word, an operator, and a constant. All WHERE clauses must specify one of the predefined WQL operators.

Following is the basic syntax for a WHERE clause appended to a SELECT statement:

```
SELECT CIM instance FROM CIM classWHERE conditional_expression
```

The conditional expression in a WHERE clause takes the following form:

property operator constant

The following subclasses of the `QueryExp` class manipulate particular types of conditional expressions in the WHERE clause:

- `AndQueryExp`
- `BinaryRelQueryExp`
- `NotQueryExp`
- `OrQueryExp`

The conditional expression in the WHERE clause is represented by the `QueryExp` class. The `QueryExp` class returns only the top level of the query expression tree. The provider can then use methods within that class to get branches down the query expression tree.

Using the Canonize Methods

The following methods are useful for providers that pass the WQL query string to another entity that parses the string:

- `canonicalizeDOC` - Canonizes the expression into a Disjunction of Conjunctions form. (OR of ANDed comparison expressions). This enables handling of the expression as a List of Lists rather than a tree form, enabling ease of evaluation. For example: (x > 5 and y > 6) or (y > 6 and z=7)
- `canonicalizeCOD` - Canonizes the expression into a Conjunction of Disjunctions form. (AND of Ored comparison expressions). This enables handling of the expression as a List of Lists rather than a tree form, enabling ease of evaluation. For example:

$(x > 5 \text{ or } y > 6) \text{ and } (y > 6 \text{ or } z=7)$

Writing a Provider that Parses WQL Query Strings

The general procedure for writing a provider that parses WQL queries using the Query APIs follows.

▼ How To Write A Provider that Parses WQL Query Strings

1. Initialize the WQL parser:

```
/* Read query string passed to execQuery from the CIM Object
   Manager into an input data stream. */
ByteArrayInputStream in = new ByteArrayInputStream(query.getBytes());

/* Initialize the parser with the input data stream. */
WQLParser parser = new WQLParser(in);
```

2. Create a vector to store the result of the query:

```
Vector result = new Vector();
```

3. Get the select expression from the query:

```
/* querySpecification returns the WQL expression from the parser.
   (SelectExp)parser casts the WQL expression to a select expression. */
SelectExp q = (SelectExp)parser.querySpecification();
```

4. Get the select list from the select expression:

```
/* Use the SelectList method in the SelectExp class
   to return the select list. The select list is the list
   of attributes, or CIM properties. */
SelectList attrs = q.getSelectList();
```

5. Get the from clause:

```
/* Use the getFromClause method in the SelectExp class
   to return the From clause. Cast the From clause to a
   Non Join Expression, a table that represents a single
   CIM class. */
NonJoinExp from = (NonJoinExp)q.getFromClause();
```

6. Use the `enumInstances` method to return a deep enumeration of the class:

```
/* Returns all instances, including inherited and local properties,
   belonging to the specified class (cc). */
Vector v = new Vector();
v = enumInstances(op, true, cc, true);
...
```

7. Iterate through the instances in the enumeration, matching the query expression and select list to each instance:

```
/* Test whether the query expression in the WHERE
   clause matches the CIM instance. Apply the select
   list to the CIM instance and add any instance that
   matches the select list (list of CIM properties)
   to the result. */
for (int i = 0; i < v.size(); i++) {
    if ((where == null) || // If there is a WHERE clause
        (where.apply((CIMInstance)v.elementAt(i)) == true)) {
        result.addElement(attrs.apply((CIMInstance)v.elementAt(i)));
    }
    ...
}
```

8. Return the query result:

```
return result;
```

Implementing the `execQuery` Method

The following sample program 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, does 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 5-6 Provider that Implements the `execQuery` Method

```
/*
 * 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
 */
```

EXAMPLE 5-6 Provider that Implements the `execQuery` Method (Continued)

```
* @author      Sun Microsystems, Inc.
*/
public CIMInstance[] execQuery(CIMObjectPath op,
                               String query,
                               String ql,
                               CIMClass cc)
    throws CIMException {

    ByteArrayInputStream in = new ByteArrayInputStream(query.getBytes());
    WQLParser parser = new WQLParser(in);
    Vector result = new Vector();
    try {
        SelectExp q = (SelectExp)parser.querySpecification();
        SelectList attrs = q.getSelectedList();
        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]));
            }
        }
    } catch (Exception e) {
        throw new CIMException(CIMException.CIM_ERR_FAILED, e.toString());
    }
    return (CIMInstance[])result.toArray();
} // execQuery
}
```


Handling CIM Events

This chapter describes CIM Indications and how they are used to communicate the occurrences of events, and the classes that enable clients to subscribe to CIM Indications. This chapter includes the following topics:

- “The CIM Event Model” on page 143
- “Creating a Subscription” on page 146
- “Generating an Event Indication” on page 151

Tip – For more in-depth information on the CIM Event Model, see the Distributed Management Task Force white paper at <http://www.dmtf.org/education/whitepapers.php>.

The CIM Event Model

Tip – The CIM Event API is located at </usr/sadm/lib/wbem/doc/javax/wbem/client/CIMEvent.html>.

An *event* is a real world occurrence, and an *indication* is an object that is created as a result of the occurrence of an event. It is important to distinguish between the event itself and the notification of the event, the *indication*. In CIM, events are not published; indications are published.

An indication is a subtype of a class that has an association with zero or more *triggers* (the description of the change) that can create instances of the class `Indication`. The WBEM implementation does not have an explicit object representing a trigger. Triggers are implied either by the operations on basic objects of the system (create,

delete, and modify on classes, instances, and namespaces) or by events in the managed environment. When an event takes place, the WBEM provider generates an indication that something happened in the system.

For example, with a *Service* class, when the service stops and a trigger is engaged, it results in an indication that serves as notification that the service stopped.

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

- Root class: `CIM_Indication`
 - Superclass: `CIM_ClassIndication`
 - Subclasses: `CIM_ClassCreation`
 - `CIM_ClassDeletion`
 - `CIM_ClassModification`
 - Superclass: `CIM_InstIndication`
 - Subclasses: `CIM_InstCreation`
 - `CIM_InstDeletion`
 - `CIM_InstModification`
 - `CIM_InstMethodCall`
 - `CIM_InstRead`
 - Superclass: `CIM_ProcessIndication`

How Indications are Generated

By default, the CIM Object Manager polls for indications of life cycle events at regular intervals. Administrators can change the event polling interval and the default polling behavior of the CIM Object Manager by editing the properties in the `cimom.properties` file. For instructions on editing the `cimom.properties` file, see the Solaris WBEM Services Administration Guide.

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 or class instance is created, modified, or deleted. 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` interface to contact the appropriate 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 the case of subscriptions pertaining to instance life cycle indication (subclasses of `CIM_InstIndication`), once the CIM Object Manager determines the CIM classes covered by the subscription, it contacts the instance providers for those CIM classes. For process indications, the appropriate provider is contacted via the `Provider` qualifier.

There are certain cases where the CIM Repository and the CIM Object Manager handle indications:

- The CIM Repository handles class indications and life cycle indications for classes that do not have providers, as well as subscriptions made to `CIM_ClassCreation`, `CIM_ClassDeletion`, `CIM_ClassModification`, `CIM_InstCreation`, `CIM_InstModification`, `CIM_InstDeletion` and `CIM_InstRead`.
- The CIM Object Manager handles `CIM_InstMethodCall`, `CIM_InstModification`, `CIM_InstDeletion` and `CIM_InstCreation` events by polling.

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 and not delegate to the CIM Object Manager, since polling has a high overhead. If the provider has to poll, the provider can delegate to the CIM Object Manager. To control the polling interval, modify the `cimom.properties`.

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. For more

information on this procedure, refer to the Solaris WBEM Services Administration Guide.

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

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).

How Subscriptions are Created

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 events in which it is interested.
- The action that the CIM Object Manager must take when the events occur.

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 for the event.

Creating a Subscription

To create a subscription, specify an instance of the `CIMListener` interface and create instances of the following classes:

- `CIM_IndicationFilter` - Defines the criteria for generating an indication and the data that is returned in the indication.
- `CIM_IndicationHandler` - Describes how to process and handle an indication. May include a destination and protocol for delivering indications.
- `CIM_IndicationSubscription` - An association that binds an event filter with an event handler.

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.

Adding a CIM Listener

To register for indications of CIM events, add an instance of the `CIMListener` interface. The CIM Object Manager generates indications for CIM events that are specified by the event filter when a client subscription is created.

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 6-1 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. An application creates an event filter by creating an instance of the `CIM_IndicationFilter` class and defining values for its properties. Event filters belong to a namespace. Each event filter works only on events that belong to the namespace to which the filter also belongs.

The `CIM_IndicationFilter` class has string properties that an application can set to uniquely identify the filter, specify a query string, and the query language to parse the query string, as shown in the following table. Currently, only the WBEM Query Language is supported.

TABLE 6-1 Properties in the `CIM_IndicationFilter` Class

Property	Description	Required/Optional
<code>SystemCreationClassName</code>	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.
<code>SystemName</code>	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.

TABLE 6-1 Properties in the CIM_IndicationFilter Class (Continued)

Property	Description	Required/Optional
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.
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 will be generated. Currently, only Level 1 WBEM Query Language expressions are supported. To learn how to construct WQL query expressions, see "Querying" on page 98.	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 event indications to be returned:

```
String filterString = "SELECT *
    FROM CIM_InstCreation WHERE sourceInstance
    is 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 to parse the query string.

Currently, only the WBEM Query Language can be used to part query strings.

```

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 from the `cimfilter` instance, called `filter`, and store it in the CIM Object Manager Repository:

```

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

```

Creating an Event Handler

An event handler is an instance of a `CIM_IndicationHandler` class. The CIM Event MOF defines a `CIM_IndicationHandlerXMLHTTP` class for describing the destination for indications to be delivered to client applications using the HTTP protocol. Event delivery to HTTP clients is not supported because HTTP delivery for events is not defined yet.

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.

An application sets the properties in the `CIM_IndicationHandler` class to uniquely name the handler and identify the UID of its owner.

TABLE 6-2 Properties in the `CIM_IndicationHandler` Class

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.
Name	The unique name of the handler.	Required. The client application must assign a unique name.

TABLE 6-2 Properties in the CIM_IndicationHandler Class (Continued)

Property	Description	Required/Optional
Owner	The name of the entity that created or maintains this handler. 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 6-2 Creating a CIM Event Handler

```
// Create an instance of the Solaris_JAVAXRMIDelivery class.
CIMClass rmidelivery = cc.getClass(new CIMObjectPath
    ("Solaris_JAVAXRMIDelivery"), false, true, true, null);

CIMInstance ci = rmidelivery.newInstance();

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

Binding an Event Filter to an Event Handler

An application binds an event filter to an event handler by creating an instance of the CIM_IndicationSubscription class. When a CIM_IndicationSubscription is created, 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 created in "To Create an Event Filter" on page 148, and defines the handler property to the *delivery* object created in Example 6-2.

EXAMPLE 6-3 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);
```

Generating an Event Indication

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

- Using the methods in the `EventProvider` interface to detect when to start and stop delivering indications of 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.

Methods in the EventProvider Interface

An event provider must implement 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 executed by an event provider.

TABLE 6-3 Methods in the `EventProvider` Interface

Method	Description
<code>activateFilter</code>	When a client creates a subscription, the CIM Object Manager calls this method to ask the provider to check for CIM events.
<code>authorizeFilter</code>	When a client creates a subscription, the CIM Object Manager calls this method to test if the specified filter expression is allowed.
<code>deActivateFilter</code>	When a client removes a subscription, the CIM Object Manager calls this method to ask the provider to deactivate the specified event filter.

TABLE 6-3 Methods in the EventProvider Interface (Continued)

Method	Description
<code>mustPoll</code>	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, and it 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 and 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`.

In order to keep track of the 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.

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.

CIM Indication Classes

Providers generate indications of CIM events by creating instances of subclasses of the `CIM_Indication` class.

The following table lists the life cycle CIM events that a provider generates.

TABLE 6-4 CIM Events Indication Classes

Event Class	Description
<code>CIM_InstCreation</code>	Notifies when a new instance is created.
<code>CIM_InstDeletion</code>	Notifies when an existing instance is deleted.
<code>CIM_InstModification</code>	Notifies when an instance is modified. The indication must include a copy of the previous instance whose change generated the indication.

▼ To Generate an Event Indication

1. Implement the `EventProvider` interface:

```
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 6-3 for each instance indication that the provider handles.

3. Create an indication listed in Table 6-4 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);
return newop;
```

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 155
- “Running the Sample Applet” on page 156
- “About the Client Sample Programs” on page 156
- “About the Provider Sample Program” on page 159

About the Sample Programs

The Solaris WBEM SDK provides a sample Java™ applet and programs installed in `/usr/demo/wbem`. You can use these samples as a basis for developing your own programs.

Note – These examples assume that `/usr/java` points to JDK 1.2 and that WBEM files are installed in the `/usr` directory.

The following samples are provided:

- Applet – List and describe the Solaris software packages that are installed on a system running Solaris WBEM Services, and connect to the CIM Object Manager running on a local or a remote system.
- Client programs – Use the client and CIM APIs to make requests to the CIM Object Manager
- Provider programs – Communicate with managed objects to access data

Running the Sample Applet

To run the applet you need the following:

- Java Development Kit (JDK) 1.2 Appletviewer, or a Java-enabled Web browser that uses JRE 1.2.2 or Java™ Plug-in 1.2.2. If you need more information on Java Plug-in, refer to the Java Plug-in documentation.
- A client machine that has network access to a system running the CIM Object Manager

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

▼ 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 using a Web Browser

- Open the `/usr/demo/wbem/applet/GetPackageInfoAp.html` file in your Web browser.

About the Client Sample Programs

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

TABLE 7-1 Client Sample Programs

Directory	Program(s)	Purpose
./batching/	<i>./TestBatch host username password classname [rmi http]</i>	Perform <code>enumerateInstanceName</code> , <code>getClassName</code> , and <code>enumerateInstances</code> in a single batching call.
./enumeration	<i>./ClientEnum host username password classname [rmi http]</i>	Enumerate classes and instances in the specified class in the default namespace <code>root\cimv2</code> on the specified host.
./events	<i>./Subscribe host username password classname</i>	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	<i>./CreateLog host root_username root_password [rmi http]</i>	Create a log record on the specified host.
	<i>./ReadLog host root_username root_password [rmi http]</i>	Read a log record on the specified host.
./misc	<i>./DeleteClass host classname root_username root_password [rmi http]</i>	Delete the specified class in the default namespace <code>root\cimv2</code> on the specified host.
	<i>./DeleteInstances host classname root_username root_password [rmi http]</i>	Delete instances of the specified class in the default namespace <code>root\cimv2</code> on the specified host.
./namespace	<i>./CreateNameSpace host parentNS childNS root_username root_password [rmi http]</i>	Connect to the CIM Object Manager as the specified user and create a namespace on the specified host.
	<i>./DeleteNameSpace host parentNS childNS root_username root_password [rmi http]</i>	Delete the specified namespace on the specified host.
	<i>./CreateQualifierType host namespace root_username root_password qualifier_type_name [rmi http]</i>	Create the specified qualifier type in the specified namespace on the specified host.
./query	<i>./ExampleQuery host username password [rmi http]</i>	Create a test class with sample instances and perform queries on that class.
	<i>./TestQuery host username password [rmi http]</i>	Perform the specified WQL query.

TABLE 7-1 Client Sample Programs (Continued)

Directory	Program(s)	Purpose
<code>./systeminfo</code>	<code>./SystemInfo host username password [rmi http]</code>	Display Solaris processor and system information for the specified host in a separate window.

Running the Client Sample Programs

You must first set up your environment before you run the client programs.

▼ How to Set Up Your Environment

- **Set your CLASSPATH environment variable to use the WBEM jar files.**

If you use the C shell:

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

If you use the Bourne shell:

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

▼ How to Run the Sample Programs

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

- **To run the sample programs use the following format:**

```
% java program_name parameters
```

EXAMPLE 7-1 Running the SystemInfo Client Program

The following example connects to *myhost* as *root* user with the *secret* password using the HTTP protocol, and runs the SystemInfo Java program:

```
% java SystemInfo myhost root secrethttp
```

About the Provider Sample Program

The provider sample program returns system properties and prints the string, “Hello World”. The program calls native C methods to execute the code and return the values to the provider.

Note – For detailed information on writing and integrating Java programs with native methods, visit the Java Web page at <http://www.javasoft.com/docs/books/tutorial/native1.1/TOC.html>.

The program files are located in `/usr/demo/wbem/provider`, and are described in the following table.

TABLE 7-2 Provider Sample Files

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

TABLE 7-2 Provider Sample Files (Continued)

File Name	Purpose
Native_Example.h	Machine-generated header file for 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.

Running the Provider Sample Program

The sample provider program, `NativeProvider`, enumerates instances and gets properties for instances of the `Native_Example` class. You can use CIM WorkShop to view this class and its instances.

▼ How To Run the `NativeProvider` Program

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

- Using the C shell:

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

Using the Bourne shell:

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

Note – If 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.

2. **Copy the shared library files 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 user.**

5. **Stop the CIM Object Manager:**

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


6. Start the CIM Object Manager:

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

7. Compile Native_Example.mof to load the Native_Example class in the CIM Object Manager and identify NativeProvider as its provider:

```
% mofcomp Native_Example.mof
```

8. Start CIM WorkShop:

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

9. In the CIM WorkShop Toolbar, click the Find Class icon.

10. In the Input dialog box, type Native_Example and click OK.

The Native_Example class displays.

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 163
- “List of Error Messages” on page 164

About WBEM Error Messages

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

Parts of an Error Message

An error message consists of the following parts:

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

For example, the MOF Compiler returns the following error message:

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

Where:

- REF_REQUIRED is the *unique identifier*
- Association class CIM_Docked needs at least two refs is the *exception message*.
- CIM_Docked is a *parameter*.
- Error in line 12 indicates the *line number* in the .mof file where the error occurred.

Error Message Templates

WBEM provides error message templates in the `ErrorMessages_en.properties` file of each API. If the exception message includes parameters, the first parameter is represented as `{0}`, the second parameter is represented as `{1}`, and so on.

In this error message:

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

The following template is used:

```
REF_REQUIRED = Association class {0} needs at least
two refs.
```

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: Instances were programmed for the specified class. However, the specified class is an abstract class, and abstract classes cannot have instances.

Solution: Remove the programmed instances.

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 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, and 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 *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 of 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 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 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 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 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 a generic, and can be displayed for many different error conditions.

Solution: Because this error message is generic, many conditions can cause the message. 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 class that is missing a schema prefix.

Cause: A class was created without providing a schema prefix in front of the class name. The Common Information Model requires that all classes are provided with a schema prefix. For example, classes developed as part of the CIM Schema require a CIM prefix: CIM_Container. Classes developed as part of the Solaris Schema require a Solaris prefix: Solaris_System.

Solution: Provide the appropriate schema prefix for the class definition. Find all instances of the class missing the prefix and replace them with the class name and prefix.

CIM_ERR_INVALID_QUERY

Description: This error message uses two parameters:

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

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

Solution: Fix 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 particular superclass, but the superclass does not exist. The specified superclass may be misspelled, or a non-existent superclass name may have been accidentally specified in place of the intended superclass name. Or, the superclass and the subclass may have been interpolated:

the specified superclass actually may be a subclass of the specified 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 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 is low on memory.

Solution: You may have to 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 may be misspelled, or a non-existent class name may 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 may 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 typing and spelling are correct.

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: The CLASS_REFERENCE 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 part of an association relationship. A class can only be defined to have a reference as a property if it participates in an association relationship with another class.

Solution: Create the association relationship. Then set up the reference to the association as a property of this class.

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 an invalid password has been entered.

Solution: If you receive this message from CIM WorkShop, delete the invalid password from the Password field of the CIM WorkShop authentication dialog box and retype the password. If this error message was received from the MOF Compiler, at the system prompt, log in again and type the correct password. Ensure that you spell the password correctly.

INVALID_QUALIFIER_NAME

Description: This error message uses one parameter, {0}, which is replaced by the Managed Object Format 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 in the context of the qualifier definition.

KEY_OVERRIDE

Description: This error message uses two parameters:

- {0} is replaced by the name of the non-abstract class that is put 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 non-abstract 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} 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: No action can be taken.

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: This error message does not use parameters.

Cause: The property provider class is not found.

Solution: Ensure that the class path of the CIMOM contains the provider class parameters, the indication class for which the provider is being defined, and the name of the Java provider class.

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 class path of the CIMOM contains the provider class parameters, the indication class for which the provider is being defined, and the name of the Java provider class.

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.

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.

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, but the method of the superclass already has been overridden by a method that belongs to another subclass. The overridden method that you are trying to override does not exist in the class hierarchy because it has never 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, but it doesn't succeed because the property of the superclass already has been overridden. The property that you are trying to override does not exist in the class hierarchy.

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.

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 CIM Specification by the Distributed Management Task Force at: <http://dmtf.org/spec/cims.html>.

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 may have been misspelled or typed differently in the definition.

Solution: Define the method as an operation for the specified class. Otherwise, ensure that the method name and class name were typed 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 may have been mistyped upon login, or a user account has not been set up for the user.

Solution: Ensure that the user name is spelled and typed correctly upon login. Ensure that a user account has been set up for the user.

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 was not defined as part of the extension schema. The qualifier is required to be defined as part of the CIM Schema or 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 Distributed Management Task Force CIM Specification at: <http://www.dmtf.org/spec/cims.html>.

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 is required to be defined as part of the CIM schema or 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 Distributed Management Task Force CIM Specification at: <http://www.dmtf.org/spec/cims.html>.

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: This message is displayed when a session has been infringed upon by an intruder. The CIM Object Manager removes the session when it detects that someone is trying to maliciously change data. For information about Solaris WBEM Services security features, see "Administering Security" in *Solaris WBEM Services Administration Guide*.

Solution: Ensure that your CIM environment is secure.

NOT_HELLO

Description: This error message does not 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, indicating a security breach.

Solution: No action is available in response to this error message. For information about Solaris WBEM Services security features, see "Administering Security" in *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 Java provider class present in the class path implements the `InstanceProvider` interface. Use the following command when you declare the provider: `public Solaris implements InstanceProvider`. For information about how to implement Solaris WBEM Services providers, see Chapter 5.

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` 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`. For information about how to implement Solaris WBEM Services providers, see Chapter 5.

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`. For information about how to implement Solaris WBEM Services providers, see Chapter 5.

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, indicating a security breach.

Solution: No action is available in response to this error message. For information about Solaris WBEM Services security features, see “Administering Security” in *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.

PS_CONFIG

Description: This error message uses one parameter, {0}, which is replaced by a description of the details that cause the error to occur. The description varies depending on the type of database used for the repository and the type of situation that causes the error message.

Cause: Solaris WBEM Services requires that you run `wbemconfig(1M)` after installing the Solaris operating environment. `wbemconfig` configures the persistent store and compiles the MOF files that provide the CIM and Solaris Schema classes. If you do not run `wbemconfig` after you install Solaris WBEM Services, this error occurs. If you configure the repository after installing Solaris WBEM Services and this error occurs, you might have corrupted the database.

Solution: Run `wbemconfig`. Running `wbemconfig` is described in “Upgrading the CIM Object Manager Repository” in *Solaris WBEM Services Administration Guide*.

PS_UNAVAILABLE

Description: This error message uses one parameter, {0}, which is replaced by a message that describes why the persistent store became unavailable.

Cause: This error message is displayed when the CIM Repository is unavailable. This situation could occur if the host on which the CIM Repository resides is brought down temporarily for maintenance, or if the host on which the CIM Repository resides becomes damaged and the repository is taken down and then restored on another host.

Solution: If you receive this message while working in the CIM WorkShop, click the icon that causes the CIM WorkShop authentication dialog box to display. Then, in the Host field, type the name of another host that is running the CIM Repository and the CIM Object Manager. Type the namespace in the Namespace field, your user name and password, and log in. If you receive this message when running the MOF Compiler, type the following command to point to another host running the CIM Repository and CIM Object Manager: `mofcomp -c hostname` where `mofcomp` is the command to start the MOF Compiler, `-c` is the parameter that enables you to specify a host computer running the CIM Object Manager, and `hostname` is the name of the specified computer.

QUALIFIER_UNOVERRIDABLE

Description: This error message uses two parameters:

- {0} parameter is replaced by the name of the qualifier that is set with the `DisableOverride` flavor.

- {1} parameter is replaced by the name of the qualifier that is set to be disabled by {0}.

Cause: The ability of the specified qualifier to override another qualifier is disabled because the flavor of the specified qualifier has been set to `DisableOverride` or `Override=False`.

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 specified to participate in an association relationship.

Cause: A class was set up to participate in an association, but no references were cited. The rules of the Common Information Model specify that an association must contain two or more references.

Solution: Set up the references to the class. Then set up the association.

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 requirements of the CIM Specification. For example, the `[READ]` qualifier is defined in the CIM Specification to modify a Property. 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 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 section, “1.Qualifiers” of the Distributed Management Task Force CIM Specification at the following URL:http://www.dmtf.org/spec/cim_spec_v20 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.

SIGNATURE_ERROR

Description: This error message does not use parameters.

Cause: This message is displayed when a message is accidentally or maliciously corrupted. It differs from the checksum error in that the message has a valid checksum, but the signature cannot be verified by the public key of the client. This protection ensures that even though the session key has been compromised, only the initial client which created the session is authenticated.

Solution: No action is provided for this message, which is displayed when a session has been infringed upon by an intruder. For information about Solaris WBEM Services security features, see "Administering Security" in *Solaris WBEM Services Administration Guide*.

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 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. Ensure that no typing errors were made. 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 upgraded version of Solaris WBEM Services does not support the current CIM Object Manager.

Solution: Install the supported version.

Glossary

This Glossary defines terms used in the Solaris WBEM documentation. Many of these terms are familiar to developers, but have a new or altered meaning in the WBEM environment.

Tip – For an expanded glossary, refer to the Distributed Management Task Force Glossary at <http://www.dmtf.org/education/cimtutorial/reference/glossary.php>

alias	A symbolic reference in either a class or instance declaration to an object located elsewhere in a MOF file. Alias names follow the same rules as instance and class names. Aliases are typically used as shortcuts to lengthy paths.
aggregation relationship	A relationship in which one entity is made up of the aggregation of some number of other entities.
association class	A class that describes a relationship between two classes or between instances of two classes. The properties of an association class include pointers, or references, to the two classes or instances. All WBEM classes can be included in one or more associations.
Backus-Naur Form (BNF)	A metalanguage that specifies the syntax of programming languages.
cardinality	The number of values that may apply to an attribute for a given entity.
class	A collection or set of objects that have similar properties and fulfill similar purposes.
CIM Object Manager Repository	A central storage area managed by the Common Information Model Object Manager (CIM Object Manager). This repository contains the definitions of classes and instances that represent managed objects and the relationships among them.
CIM Schema	A collection of class definitions used to represent managed objects that occur in every management environment.

See also core model, common model, and extension schema.

The CIM is divided into the metamodel and the standard schema. The metamodel describes what types of entities make up the schema. It also defines how these entities can be combined into objects that represent managed objects.

common model

The second layer of the CIM schema, which includes a series of domain-specific but platform-independent classes. The domains are systems, networks, applications, and other management-related data. The common model is derived from the core model.

See also extension schema.

core model

The first layer of the CIM schema, which includes the top-level classes and their properties and associations. The core model is both domain- and platform-independent.

See also common model and extension schema.

**Distributed
Management Task Force
(DMTF)**

An industry-wide consortium committed to making personal computers easier to use, understand, configure, and manage.

domain

The class to which a property or method belongs. For example, if status is a property of Logical Device, it is said to belong to the Logical Device domain.

dynamic class

A class whose definition is supplied by a provider at runtime as needed. Dynamic classes are used to represent provider-specific managed objects and are not stored permanently in the CIM Object Manager Repository. Instead, the provider responsible for a dynamic class stores information about its location. When an application requests a dynamic class, the CIM Object Manager locates the provider and forwards the request. Dynamic classes support only dynamic instances.

dynamic instances

An instance that is supplied by a provider when the need arises and is not stored in the CIM Object Manager Repository. Dynamic instances can be provided for either static or dynamic classes. Supporting instances of a class dynamically allows a provider to always supply up-to-the-minute property values.

enumeration

Java term for getting a list of objects. Java provides an `Enumeration` interface that has methods for enumerating a list of objects. An individual object on this list to be enumerated is called an element.

extension schema

The third layer of the CIM Schema, which includes platform-specific extensions of the CIM Schema such as Solaris and UNIX.

See also common model and core model.

flavor	<i>See</i> qualifier flavor.
indication	An operation executed as a result of some action such as the creation, modification, or deletion of an instance, access to an instance, or modification or access to a property. Indications can also result from the passage of a specified period of time. An indication typically results in an event.
inheritance	The relationship that describes how classes and instances are derived from parent classes or superclasses. A class can spawn a new subclass, also called a child class. A subclass contains all the methods and properties of its parent class. Inheritance is one of the features that allows WBEM classes to function as templates for actual managed objects in the WBEM environment.
instance	A representation of a managed object that belongs to a particular class, or a particular occurrence of an event. Instances contain actual data.
instance provider	A type of provider that supports instances of system- and property-specific classes. Instance providers can support data retrieval, modification, deletion, and enumeration. Instance providers can also invoke methods. <i>See also</i> property provider.
interface class	The class used to access a set of objects. The interface class can be an abstract class representing the scope of an enumeration. <i>See also</i> enumeration and scope.
Interface Definition Language (IDL)	A generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.
key	A property that is used to provide a unique identifier for an instance of a class. Key properties are marked with the Key qualifier.
Key qualifier	A qualifier that must be attached to every property in a class that serves as part of the key for that class.
managed object	A hardware or software component that is represented as an instance of the CIM class. Information about managed objects is supplied by providers as well as the CIM Object Manager. <i>See also</i> managed resource.
Managed Object Format (MOF)	A compiled language for defining classes and instances. The MOF compiler (mofc) compiles .mof text files into Java classes and adds the data to the CIM Object Manager Repository. MOF eliminates the need to write code, thus providing a simple and fast technique for modifying the CIM Object Manager Repository.

managed resource	<p>A hardware or software component that can be managed by a management application. Hard disks, CPUs, and operating systems are examples of managed resources. Managed resources are described in WBEM classes.</p> <p><i>See also</i> managed object.</p>
management application	<p>An application or service that uses information originating from one or more managed objects in a managed environment. Management applications retrieve this information through calls to the CIM Object Manager API from the CIM Object Manager and from providers.</p>
management information base	<p>A database of managed objects.</p>
metamodel	<p>A CIM component that describes the entities and relationships representing managed objects. For example, classes, instances, and associations are included in the metamodel.</p>
metaschema	<p>A formal definition of the Common Information Model, which defines the terms used to express the model, its usage, and its semantics.</p>
method	<p>A function describing the behavior of a class. Including a method in a class does not guarantee an implementation of the method.</p>
MOF file	<p>A text file that contains definitions of classes and instances using the Managed Object Format (MOF) language.</p>
Named Element	<p>An entity that can be expressed as an object in the metaschema.</p>
namespace	<p>A directory-like structure that can contain classes, instances, and other namespaces.</p>
object path	<p>A formatted string used to access namespaces, classes, and instances. Each object on the system has a unique path which identifies it locally or over the network. Object paths are conceptually similar to Universal Resource Locators (URLs).</p>
override	<p>Indicates that the property, method, or reference in the derived class overrides the similar construct in the parent class in the inheritance tree or in the specified parent class.</p>
polymorphism	<p>The ability to alter methods and properties in a derived class without changing their names or altering interfaces. For example, a subclass can redefine the implementation of a method or property inherited from its superclass. The property or method is thereby redefined even if the superclass is used as the interface class.</p> <p>Thus, the LogicalDevice class can define the variable status as a string, and can return the values "on" or "off." The Modem subclass of LogicalDevice can redefine (override) status by returning "on," "off," and "connected." If all LogicalDevices are enumerated, any</p>

	LogicalDevice that happens to be a modem can return the value "connected" for the status property.
property	A value used to characterize the instances of a class. Property names cannot begin with a digit and cannot contain white space. Property values must have a valid Managed Object Format (MOF) data type.
property provider	A program that communicates with managed objects to access data and event notifications from a variety of sources, such as the Solaris operating environment or a Simple Network Management Protocol (SNMP) SNMP device. Providers forward this information to the CIM Object Manager for integration and interpretation.
qualifier	A modifier containing information that describes a class, an instance, a property, a method, or a parameter. The three categories of qualifiers are: those defined by the Common Information Model (CIM), those defined by WBEM (standard qualifiers), and those defined by developers. Standard qualifiers are attached automatically by the CIM Object Manager.
qualifier flavor	An attribute of a CIM qualifier that governs the use of a qualifier. WBEM 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.
range	A class that is referenced by a reference property.
reference	A special string property type that is marked with the reference qualifier, indicating that it is a pointer to other instances.
required property	A property that must have a value.
schema	A collection of class definitions that describe managed objects in a particular environment.
scope	An attribute of a CIM qualifier that indicates which CIM elements can use the qualifier. Scope can only be defined in the Qualifier Type declaration; it cannot be changed in a qualifier.
selective inheritance	The ability of a descendant class to drop or override the properties of an ancestral class.
Simple Network Management Protocol (SNMP)	A protocol of the Internet reference model used for network management.
singleton class	A WBEM class that supports only a single instance.
Solaris Schema	A Sun extension to the CIM Schema that contains definitions of classes and instances to represent managed objects that exist in a typical Solaris operating environment.
standard schema	A common conceptual framework for organizing and relating the various classes representing the current operational state of a system,

network, or application. The standard schema is defined by the Distributed Management Task Force (DMTF) in the Common Information Model (CIM).

static class	A WBEM class whose definition is persistent. The definition is stored in the CIM Object Manager Repository until it is explicitly deleted. The CIM Object Manager can provide definitions of static classes without the help of a provider. Static classes can support either static or dynamic instances.
static instance	An instance that is persistently stored in the CIM Object Manager Repository.
subclass	A class that is derived from a superclass. The subclass inherits all features of its superclass, but can add new features or redefine existing ones.
subschema	A part of a schema owned by a particular organization. The Win32 and Solaris Schemas are examples of subschemas.
superclass	The class from which a subclass inherits.
transitive dependency	In a relation having at least three attributes R (A, B, C), the situation in which A determines B, B determines C, but B does not determine A.
trigger	A recognition of a state change (such as create, delete, update, or access) of a class instance, and update or access of a property. The WBEM implementation does not have an explicit object representing a trigger. Triggers are implied either by the operations on basic objects of the system (create, delete, and modify on classes, instances and namespaces) or by events in the managed environment.
Unified Modeling Language (UML)	A notation language used to express a software system using boxes and lines to represent objects and relationships.
Unicode	A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.
UTF-8	An 8-bit transformation format that may also serve as a transformation format for Unicode character data.
virtual function table (VTBL)	A table of function pointers, such as an implementation of a class. The pointers in the VTBL point to the members of the interfaces that an object supports.
Win32 Schema	A Microsoft extension to the CIM Schema that contains definitions of classes and instances to represent managed objects that exist in a typical Win32 environment.

Index

A

- application programming interfaces (APIs)
 - calling methods, 109
 - connecting to CIM Object Manager with
 - default namespace, 84
 - creating a namespace, 113
 - creating instances, 86
 - deleting a class, 117
 - deleting instances, 86
 - enumerating classes, 96
 - enumerating namespaces, 94
 - example program, 80
 - exception handling, 111
 - getting CIM qualifiers, 119
 - getting properties, 90
 - overview, 75
 - programming tasks, 82
 - provider, 123
 - retrieving classes, 110
 - setting CIM qualifiers, 120
 - setting instances, 92
 - specifying a namespace, 84

B

- base class
 - creating, 115

C

- CIM Object Manager
 - connecting to, 83
 - connecting to default namespace, 84
 - error messages, 163
 - how it uses providers, 123
 - registering a provider, 132
- CIM qualifiers
 - getting, 119
 - setting, 120
- CIM Schema, 26
- CIM Workshop
 - adding classes, 64
- CIM WorkShop
 - displaying and creating instances, 70
 - interface, 36
 - starting, 55
 - working in namespaces, 59
- class
 - `deleteClass`, 117
 - deleting, 117
 - enumerating, 96
 - `newInstance`, 116
 - retrieving, 110
- class definition
 - retrieving, 82
- classes
 - `CIMClass`, 115
 - creating, 115
 - deleting, 117
 - retrieving definition of, 82

- client session
 - opening, 82
- Common Information Model
 - basic terms
 - association, 25
 - extension schemas, 31
 - schema, 25
- Core Model
 - dependencies, 29
 - system classes, 27

D

- default namespace, 59, 82
- Distributed Management Task Force, 21
- dynamic data, 123

E

- Error messages, 163
- error messages
 - handling, 82
- example programs
 - running, 158
- examples
 - calling a method, 109
 - connecting to CIM Object Manager, 83
 - creating a namespace, 113
 - creating an instance, 86
 - deleting a class, 117
 - deleting an instance, 86
 - enumerating classes, 96
 - enumerating namespaces, 94
 - error message, 164
 - getting a property, 90
 - getting CIM qualifiers, 119
 - implementing a property provider, 127
 - Java output, 80
 - retrieving a class, 110
 - setting CIM qualifiers, 120
 - setting instances, 92
 - specifying a namespace, 84
- exception handling, 111

- exceptions, *See* error messages

H

- host
 - moving to another, 61

I

- instance
 - creating, 85
 - deleting, 86
 - getting and setting, 88
 - type of provider, 124
- instances
 - in CIM WorkShop, 70

J

- Java
 - creating instances, 86
 - deleting instances, 86
 - getting properties, 90
 - integrating Java programs with native methods, 130, 159
 - Java Native Interface (JNI), 130
 - setting instances, 92
 - Solaris WBEM SDK example programs, 155
 - specifying a namespace, 84

M

- Managed Object Format
 - creating base classes, 115
 - description, 32
- method
 - calling, 82
 - deleteInstance, 86
 - deleting a namespace, 113
 - enumNameSpace, 94
 - getClass, 110
 - getInstance, 88
 - getProperty, 90, 91

method (*continued*)
 getPropertyValue, 127
 invokeMethod, 109
 type of provider, 124
Methods
 calling, 109
MOF Compiler
 error checking, 112

N

namespace
 connecting to default, 84
 creating, 113
 default, 83, 112
 enumerating, 94
 refreshing a, 62
namespaces
 creating, 82

O

object
 enumerating, 82

P

property
 getting, 90, 91
 type of provider, 124
provider
 functions, 124
 implementing a Property Provider, 127
 interfaces, 125
 registering with the CIM Object
 Manager, 133
 types, 124
 writing a native provider, 130

Q

qualifier
 definition, 119

qualifier (*continued*)
 example type declaration, 119
 key, 115
Query
 executing, 57

S

schema
 CIM Schema, 26
security namespace, 82
single provider, 124
Solaris WBEM SDK
 example program, 80
 programming tasks, 82
Solaris WBEM SDK error messages, *See* error
 messages
Solaris WBEM SDK example programs, *See*
 example programs
Solaris WBEM Services error messages, *See* error
 messages

T

technology-specific schemas, 29

U

Uniform Modeling Language, 23

W

WBEM
 application programming interfaces
 (APIs), 75
 definition, 21
workshop
 See CIM WorkShop

