Contents

Preface ................................................................................................................................................... 15

1 Overview of Solaris Web-Based Enterprise Management ............................................................ 21
   About Web-Based Enterprise Management .................................................................................. 21
   About the Common Information Model ...................................................................................... 22
   About Solaris WBEM Services ..................................................................................................... 22
      Software Components ............................................................................................................. 23
      CIM Object Manager ............................................................................................................... 26
   Managed Object Format Compiler ............................................................................................. 26
   Solaris Platform Schema .......................................................................................................... 27
   Solaris WBEM SDK .................................................................................................................. 27
   Using CIM Workshop to Develop WBEM Applications ............................................................. 28
      CIM Workshop Documentation ............................................................................................... 29
      Running CIM Workshop ........................................................................................................ 29

2 Using the CIM Object Manager .................................................................................................. 31
   About the CIM Object Manager ................................................................................................. 31
   init.wbem Command .................................................................................................................. 32
      Solaris Management Console Server ...................................................................................... 32
      System Booting ...................................................................................................................... 33
   Stopping and Restarting the CIM Object Manager .................................................................... 33
      ▼ How to Stop the CIM Object Manager .............................................................................. 33
      ▼ How to Restart the CIM Object Manager ........................................................................... 33
   Upgrading the CIM Object Manager Repository ...................................................................... 33
      ▼ How to Recompile Your MOF Files .................................................................................... 34
      ▼ How to Merge WBEM Data ............................................................................................... 35
   Exception Messages .................................................................................................................. 35
## 3 Using the Sample Programs

- **About the Sample Programs** ................................................................. 37
- **Sample Applet** ................................................................................ 38
  - ▼ How to Run the Sample Applet Using Appletviewer ............................. 38
  - ▼ How to Run the Sample Applet in a Web Browser ............................ 38
- **Sample Client Programs** .................................................................. 38
  - Running the Sample Client Programs .................................................. 39
- **Sample Provider Programs** ............................................................... 40
  - ▼ How to Run the Sample Provider Programs ................................... 41

## 4 Writing a Client Program

- **Client API Overview** ......................................................................... 43
  - Sequence of a Client Application ......................................................... 43
- **Opening and Closing a Client Connection** ....................................... 44
  - **About Name Spaces** .................................................................... 44
  - Opening a Client Connection ............................................................. 45
  - Closing a Client Connection .............................................................. 46
- **Performing Basic Client Operations** .................................................. 46
  - Creating an Instance ......................................................................... 46
  - Deleting an Instance ........................................................................ 47
  - Getting and Setting Instances ............................................................ 49
  - Getting and Setting Properties .......................................................... 50
  - Enumerating Objects ........................................................................ 51
  - Creating Associations ....................................................................... 56
  - Calling Methods ................................................................................ 60
  - Retrieving Class Definitions .............................................................. 61
  - Handling Exceptions ......................................................................... 62
  - Creating a Name Space ..................................................................... 62
  - Deleting a Name Space .................................................................... 63
  - Creating a Base Class ....................................................................... 63
  - Deleting a Class ................................................................................ 64
  - Setting Access Control ..................................................................... 65
    - Solaris_UserAcl Class ..................................................................... 65
    - Solaris_NameSpaceAcl Class ......................................................... 67
  - Working With Qualifiers and Qualifier Types ..................................... 68
A Solaris Platform Schema ................................................................. 159
Solaris Platform Schema Files ............................................................ 160
Solaris_Acl.mof File ........................................................................ 161
Solaris_Application.mof File .............................................................. 161
Solaris_CIMOM.mof File .................................................................. 162
Solaris_Core.mof File ..................................................................... 162
Solaris_Device.mof File ................................................................... 163
Solaris_DMGT1.0.mof File ................................................................. 164
Solaris_Event.mof File ................................................................... 164
Solaris_Network.mof File ................................................................. 165
Solaris_Performance.mof File ........................................................... 165
Solaris_Project.mof File ................................................................ 165
Solaris_Schema.mof File ................................................................ 166
Solaris_SNMP.mof File .................................................................. 166
Solaris_System.mof File ................................................................ 166
Solaris_Users.mof File .................................................................. 167
Solaris_VM2.0.mof File .................................................................. 167
WBEMServices.mof File .................................................................. 169

Index .................................................................................................. 171
Figures

FIGURE 1–1  Solaris WBEM Services Architecture ......................................................... 24
FIGURE 4–1  TeacherStudent Association 1 ................................................................. 56
FIGURE 4–2  TeacherStudent Association 2 ................................................................. 57
Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1-1</td>
<td>Solaris WBEM APIs</td>
<td>28</td>
</tr>
<tr>
<td>TABLE 2-1</td>
<td>Determining Whether to Recompile or Merge WBEM Data</td>
<td>34</td>
</tr>
<tr>
<td>TABLE 3-1</td>
<td>Sample Client Programs</td>
<td>38</td>
</tr>
<tr>
<td>TABLE 3-2</td>
<td>Sample Provider Programs</td>
<td>40</td>
</tr>
<tr>
<td>TABLE 4-1</td>
<td>Enumerating Objects</td>
<td>51</td>
</tr>
<tr>
<td>TABLE 4-2</td>
<td>Association Methods</td>
<td>56</td>
</tr>
<tr>
<td>TABLE 4-3</td>
<td>TeacherStudent Methods</td>
<td>58</td>
</tr>
<tr>
<td>TABLE 4-4</td>
<td>invokeMethod Parameters</td>
<td>60</td>
</tr>
<tr>
<td>TABLE 4-5</td>
<td>CIM_Indication Class Structure</td>
<td>72</td>
</tr>
<tr>
<td>TABLE 4-6</td>
<td>CIM_IndicationFilter Properties</td>
<td>75</td>
</tr>
<tr>
<td>TABLE 4-7</td>
<td>CIM_IndicationHandler Properties</td>
<td>77</td>
</tr>
<tr>
<td>TABLE 4-8</td>
<td>Log Message Elements</td>
<td>79</td>
</tr>
<tr>
<td>TABLE 5-1</td>
<td>Mapping of SQL Concepts to WQL</td>
<td>85</td>
</tr>
<tr>
<td>TABLE 5-2</td>
<td>Supported WQL Key Words</td>
<td>86</td>
</tr>
<tr>
<td>TABLE 5-3</td>
<td>Sample SELECT Statements</td>
<td>87</td>
</tr>
<tr>
<td>TABLE 5-4</td>
<td>WQL Operators for WHERE Clauses</td>
<td>88</td>
</tr>
<tr>
<td>TABLE 6-1</td>
<td>Provider Types</td>
<td>95</td>
</tr>
<tr>
<td>TABLE 6-2</td>
<td>EventProvider Methods</td>
<td>104</td>
</tr>
<tr>
<td>TABLE 7-1</td>
<td>MOF File Elements</td>
<td>111</td>
</tr>
<tr>
<td>TABLE 7-2</td>
<td>How CIM Elements Map to Java Elements</td>
<td>111</td>
</tr>
<tr>
<td>TABLE 7-3</td>
<td>How CIM Data Types Map to Java Data Elements</td>
<td>112</td>
</tr>
<tr>
<td>TABLE 7-4</td>
<td>Meta Qualifiers</td>
<td>113</td>
</tr>
<tr>
<td>TABLE 7-5</td>
<td>Standard Qualifiers</td>
<td>113</td>
</tr>
<tr>
<td>TABLE 7-6</td>
<td>How MOF Elements Map to Java Elements</td>
<td>114</td>
</tr>
<tr>
<td>TABLE A-1</td>
<td>Solaris Platform Schema Files</td>
<td>160</td>
</tr>
</tbody>
</table>
Examples

EXAMPLE 3–1  Running a Sample Client Program ................................................................. 40
EXAMPLE 4–1  Connecting to the Root Account ..................................................................... 45
EXAMPLE 4–2  Connecting to a User Account ......................................................................... 45
EXAMPLE 4–3  Authenticating as an RBAC Role Identity ......................................................... 46
EXAMPLE 4–4  Closing a Client Connection ............................................................................ 46
EXAMPLE 4–5  Creating an Instance .......................................................................................... 47
EXAMPLE 4–6  Deleting Instances ........................................................................................... 47
EXAMPLE 4–7  Getting and Setting Instances .......................................................................... 49
EXAMPLE 4–8  Getting a Property ............................................................................................ 50
EXAMPLE 4–9  Setting a Property ............................................................................................. 51
EXAMPLE 4–10 Enumerating Classes ........................................................................................ 52
EXAMPLE 4–11  Enumerating Classes and Instances ............................................................. 53
EXAMPLE 4–12  Enumerating Class Names .............................................................................. 54
EXAMPLE 4–13  Enumerating Name Spaces ............................................................................ 55
EXAMPLE 4–14  Passing Instances ......................................................................................... 59
EXAMPLE 4–15  Calling a Method ............................................................................................ 60
EXAMPLE 4–16  Retrieving a Class Definition ......................................................................... 61
EXAMPLE 4–17  Creating a Name Space .................................................................................. 62
EXAMPLE 4–18  Deleting a Class .............................................................................................. 64
EXAMPLE 4–19  Setting CIM Qualifiers ................................................................................... 68
EXAMPLE 4–20  Batching Example ......................................................................................... 69
EXAMPLE 4–21  Adding a CIM Listener .................................................................................... 75
EXAMPLE 4–22  Creating an Event Filter .................................................................................. 77
EXAMPLE 4–23  Creating an Event Handler .............................................................................. 78
EXAMPLE 4–24  Binding an Event Filter to an Event Handler ................................................... 78
EXAMPLE 4–25  Creating an Instance of Solaris_LogEntry .................................................... 80
EXAMPLE 4–26  Displaying a List of Log Records ...................................................................... 82
EXAMPLE 5–1  Provider That Handles Queries ....................................................................... 90
Examples

EXAMPLE 6–1  CIMInstance Provider ................................................................. 97
EXAMPLE 6–2  Method Provider .................................................................... 100
EXAMPLE 6–3  CIMAssociator Provider ......................................................... 102
EXAMPLE 6–4  Registering a Provider ............................................................ 108
EXAMPLE 7–1  Generating JavaBeans Components .................................. 115
EXAMPLE 9–1  Parts of an Error Message ..................................................... 143
Preface

The Solaris WBEM Developer’s Guide explains Common Information Model (CIM) concepts. In addition, this guide describes how to administer Web-Based Enterprise Management (WBEM) services in the Solaris Operating System (Solaris OS).

In addition, this guide describes the Solaris Web-Based Enterprise Management Software Developer’s Kit (WBEM SDK). The WBEM SDK enables developers to create standards-based applications that manage resources in the Solaris OS. 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 the following items:

- Client application programming interfaces (APIs) for describing and managing resources which use the Distributed Management Task Force (DMTF) Common Information Model (CIM)
- Provider APIs for getting and setting dynamic data on managed resources
- Sample WBEM client and provider programs
- CIM Workshop, an application that is written in the Java programming language, which you can use to create and view managed resources on a system

Note – This Solaris release supports systems that use the SPARC and x86 families of processor architectures: UltraSPARC, SPARC64, IA-32, AMD64. The supported SPARC based systems are listed in the Solaris Sun Hardware Platform Guide at http://docs.sun.com. The supported x86 based systems appear in the Solaris Hardware Compatibility List at http://www.sun.com/bigadmin/hcl. This document cites any implementation differences between the platform types.

In this document the term “x86” refers to the Intel 32-bit family of microprocessors and compatible 64-bit and 32-bit microprocessors made by AMD. For supported systems, see the Solaris Hardware Compatibility List.
Who Should Use This Book

This book is for the following types of software developers:

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

Before You Read This Book

This book requires a solid understanding of the following information:

- Object-oriented programming concepts
- The Java programming language
- Common Information Model (CIM) concepts
- Network management concepts

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

- CIM Tutorial (http://www.dmtf.org/education/cimtutorial.php), provided by the Distributed Management Task Force.

The following web sites are useful resources when working with WBEM technologies:

- Distributed Management Task Force (DMTF) – http://www.dmtf.org
  This site discusses the latest CIM developments, provides information about various working groups, and lists contact information for extending the CIM schema.
  This site contains documentation about the Unified Modeling Language (UML) and the Rose CASE tool.
How This Book Is Organized

Chapter 1, “Overview of Solaris Web-Based Enterprise Management,” introduces Web-Based Enterprise Management (WBEM), the Common Information Model (CIM), the application programming interfaces (APIs) in the Solaris WBEM SDK, and CIM Workshop.

Chapter 2, “Using the CIM Object Manager,” describes the CIM Object Manager. This chapter covers how to start and how to stop the CIM Object Manager and how to upgrade the CIM Object Manager Repository.

Chapter 3, “Using the Sample Programs,” describes the sample programs that are provided with the Solaris WBEM SDK.

Chapter 4, “Writing a Client Program,” explains how to use the client APIs to write client programs.

Chapter 5, “Writing WBEM Queries,” explains how to use the WBEM Query Language (WQL) and the Query APIs to write and handle queries.

Chapter 6, “Writing a Provider Program,” explains how to use the provider APIs to write provider programs.

Chapter 7, “Creating JavaBeans Components Using the MOF Compiler,” explains how to use the MOF compiler.

Chapter 8, “Administering Security,” describes WBEM security mechanisms and the features that the CIM Object Manager enforces.

Chapter 9, “Troubleshooting,” describes how to view log data and explains the error messages that are generated by components of the Solaris WBEM SDK.

Appendix A, “Solaris Platform Schema,” describes the MOF files that are included with the Solaris WBEM SDK.

Related Information

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

Accessing Sun Documentation Online

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

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print. For a list of documents and how to order them, see "Buy printed documentation” at http://docs.sun.com.

Typographic Conventions

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

<table>
<thead>
<tr>
<th>Typeface or Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories, and onscreen computer output</td>
<td>Edit your .login file. Use ls -a to list all files. machine_name% you have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with onscreen computer output</td>
<td>machine_name% su Password:</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Command-line placeholder: replace with a real name or value</td>
<td>The command to remove a file is rm filename.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new terms, and terms to be emphasized</td>
<td>Read Chapter 6 in the User’s Guide. Perform a patch analysis. Do not save the file. [Note that some emphasized items appear bold online.]</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Shell Prompt</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell prompt</td>
<td>machine_name%</td>
</tr>
<tr>
<td>C shell superuser prompt</td>
<td>machine_name#</td>
</tr>
<tr>
<td>Bourne shell and Korn shell prompt</td>
<td>$</td>
</tr>
<tr>
<td>Bourne shell and Korn shell superuser prompt</td>
<td>#</td>
</tr>
</tbody>
</table>
Overview of Solaris Web-Based Enterprise Management

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

- “About Web-Based Enterprise Management” on page 21
- “About the Common Information Model” on page 22
- “About Solaris WBEM Services” on page 22
- “Using CIM Workshop to Develop WBEM Applications” on page 28

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

About Web-Based Enterprise Management

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

The Distributed Management Task Force (DMTF) is an industry group that represents corporations in the computer and telecommunications industries. The DMTF 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. The approach is intended to result in cost-effective products that interoperate as flawlessly as possible.
About the Common Information Model

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

CIM consists of the following components:

- **CIM Specification** – Defines the language and methodology for integration with other management models.
- **CIM Schema** – Provides the actual model descriptions for systems, applications, local area networks, and devices. The CIM schema consists of the following 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 which are independent of a particular technology or a particular implementation. Provides a basis for the development of management applications.
  - **Extension schema** – Represents technology and platform-specific extensions to the Common Model. These schemas are specific to environments such as operating systems. For example, the Solaris schema is an extension schema. Vendors extend the model for their products by creating subclasses of objects. Applications can then transverse object instances in the standard model to manage different products in a heterogeneous environment.

About Solaris WBEM Services

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

- “CIM Object Manager” on page 26
- “Managed Object Format Compiler” on page 26
- “Solaris Platform Schema” on page 27
- “Solaris WBEM SDK” on page 27

Solaris WBEM Services software provides WBEM services in the Solaris OS, including secure access and manipulation of management data. The software includes a Solaris platform provider that enables management applications to access information about managed resources such as devices and software in the Solaris OS.

The CIMOM accepts connections from management applications that use either the Remote Method Invocation (RMI) protocol or the XML over HTTP protocol. The CIMOM provides the following services to connected clients:
- **Management services** – These services are in the form of a CIMOM. The CIMOM checks the semantics and syntax of CIM data. The CIMOM also distributes data between applications, the CIM Object Manager Repository, and managed resources.

- **Security services** – Specify these services for WBEM through the Solaris Management Console User tool. These services are described in *System Administration Guide: Security Services*.

- **Sun WBEM User Manager** – Use this tool to establish an access control list (ACL) for a specific name space on the WBEM server. ACL-based security is uniquely provided by Solaris WBEM Services. Sun WBEM User Manager performs the following tasks:
  - Enables you to add and delete authorized users
  - Set access privileges for authorized users
  - Manage user authentication and access to CIM objects on a WBEM-enabled system

- **Logging services** – Consist of classes that developers can use to create applications that dynamically record and retrieve event data. Administrators use this data to track and determine the cause of events. Logging services are described in more detail in Chapter 9, “Troubleshooting.”

- **XML services** – Convert XML data into CIM classes, enabling XML/HTTP-based WBEM clients to communicate with the CIM Object Manager.

Once connected to a WBEM-enabled system, WBEM clients can request WBEM operations to:

- Create, view, and delete CIM classes and instances
- Query for properties that have a specified value
- Enumerate instances or classes in a specified class hierarchy

### Software Components

Solaris WBEM Services software consists of three software components: Application, Management, and Provider. These components interact with the operating system and with hardware. The following figure shows the software components and how these components interact.
**Application layer** – WBEM clients process and display data from managed resources. Solaris WBEM Services include the following applications.
- **Sun WBEM User Manager and Solaris Management Console User tool** – Applications that enable system administrators to add and delete authorized users and to set these users' access privileges to managed resources.

- **Solaris Management Console Log Viewer** – An application that displays log files. A user can view details of a log record, including the name of the user who issued a logged command and the client computer on which a logged event occurred.

- **Managed Object Format (MOF) compiler** – A program that parses a file containing MOF statements, converts the classes and instances defined in the file to Java classes, and then adds the Java classes to the CIM Object Manager Repository, a central storage area for management data.

  MOF is a language for defining CIM classes and instances. MOF files are ASCII text files that use the MOF language to describe CIM objects. A CIM object is a representation, or model, of a managed resource, such as a printer, disk drive, or CPU. MOF files are located in `/usr/sadm/mof`.

  Many sites store information about managed resources in MOF files. Because MOF can be converted to Java, applications that can run on any system with a Java virtual machine can interpret and exchange this information. You can also use the `mofcomp` command to compile MOF files at any time after installation. MOF is described on the DMTF web page at [http://www.dmtf.org](http://www.dmtf.org).

- **Management layer** – Components at this layer provide services to connected WBEM clients.

  - **Common Information Model (CIM) Object Manager** – Software that manages CIM objects on a WBEM system. CIM objects are stored internally as Java classes. The CIM Object Manager transfers information between WBEM clients, the CIM Object Manager Repository, and managed resources.

  - **CIM Object Manager Repository** – Central storage area for CIM class and instance definitions.

  - **Client and CIM application programming interfaces (APIs)** – WBEM client applications use these Java interfaces to request operations from the CIM Object Manager. Such operations include creating or viewing classes or instances of managed resources.

  - **Provider interfaces** – Providers use these interfaces to transfer information about managed resources to the CIM Object Manager. The CIM Object Manager uses the provider interfaces to transfer information to locally installed providers.

  - **Provider layer** – Providers act as intermediaries between the CIM Object Manager and one or more managed resources. When the CIMOM receives a request from a WBEM client for data that is not available from the CIM Object Manager Repository, the CIMOM forwards the request to the appropriate provider.

  - **Solaris providers** – Provide the CIM Object Manager with instances of managed resources in the Solaris OS. Providers get and set information on managed devices. A native provider is a machine-specific program that is written to run on a managed
device. For example, a provider that accesses data on a system running the Solaris OS probably includes C functions to query that system. The Java Native Interface is part of the JDK software. By writing programs using the Java Native Interface, you ensure that your code is portable across all platforms. The Java Native Interface enables Java code that runs within a Java virtual machine to operate with applications and libraries that are written in other languages, such as C, C++, and assembly.

- **Solaris platform schema** – A collection of classes that describes managed objects in the Solaris OS. The CIM schema and Solaris platform schema classes are stored in the CIM Object Manager Repository. The CIM schema is a collection of class definitions used to represent managed objects that occur in every management environment. The Solaris platform schema is a collection of class definitions that extend the CIM schema to represent managed objects in a typical Solaris OS. Users can also use the MOF compiler (mofcomp) to add CIM schema, Solaris platform schema, or other classes to the CIM Object Manager Repository.

- **Operating system layer** – The Solaris platform providers enable management applications to access information about managed resources such as devices and software, in the Solaris OS.

- **Hardware layer** – A management client can access management data on any supported Solaris platform.

## CIM Object Manager

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

WBEM client applications contact the CIM Object Manager to establish a connection. This connection is used 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. The client can use that reference to request services and perform operations.

## Managed Object Format Compiler

You use the Managed Object Format (MOF) language to specify CIM schema. You define classes and instances using ASCII text, and place those classes in a file that you submit to the MOF compiler, mofcomp. The MOF compiler parses the file to add the classes and instances defined in the file to the CIM Object Manager repository. See Chapter 7, "Creating JavaBeans Components Using the MOF Compiler," for information on how to use the MOF compiler to automatically generate JavaBeans components from MOF files.
Because you can convert MOF to Java, applications developed in MOF can run on any system or in any environment that supports the Java platform.

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

Solaris Platform Schema

The Solaris platform schema is an extension schema of the Common Model. The Solaris platform schema specifically describes managed objects running in the Solaris OS.

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

Note – The CIM schema and Solaris platform schema are installed at file:/usr/sadm/lib/wbem/doc/mofhtml/index.html.

Solaris WBEM SDK

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

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

Note – The Solaris WBEM application programming interface (API) documentation is in Javadoc format and is installed at file:/usr/sadm/lib/wbem/doc/index.html during a Solaris OS installation.

The Solaris WBEM APIs are described in the following table.
### Solaris WBEM APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIM</td>
<td>javax.wbem.cim</td>
<td>Includes common classes and methods that represent the basic CIM elements. The CIM APIs create objects on the local system.</td>
</tr>
<tr>
<td>Client</td>
<td>javax.wbem.client</td>
<td>Applications use the CIMClient class to connect to the CIM Object Manager. Applications use the other classes and methods to transfer data to and from the CIM Object Manager. The Batching APIs, a subset of the Client APIs, enable clients to batch multiple requests in one remote call. This capability reduces the delay introduced by multiple remote message exchanges.</td>
</tr>
<tr>
<td>Provider</td>
<td>javax.wbem.provider</td>
<td>The CIM Object Manager uses these APIs to pass application requests for dynamic data to providers.</td>
</tr>
<tr>
<td>Query</td>
<td>javax.wbem.query</td>
<td>Contains classes and methods that you use to formulate and manipulate queries by using the WBEM Query Language (WQL).</td>
</tr>
</tbody>
</table>

### Using CIM Workshop to Develop WBEM Applications

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

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

**Note** – CIM guidelines prevent you from modifying the properties, methods, and qualifiers of CIM schema and Solaris platform schema classes. You also cannot change the values of inherited properties, methods, and qualifiers.
CIM Workshop Documentation

CIM Workshop has context-sensitive online help for every dialog box except for the main window. When you click the interface components, the appropriate help text displays in the Information pane on the left side of the dialog box.

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

Running CIM Workshop

The default behavior of CIM Workshop is to:

- Connect to a CIMOM running on the local host
- In the root\cimv2 default name space
- Using the Remote Method Invocation (RMI) protocol

You can also point to a remote host that is running the CIM Object Manager.

▼ How to Start CIM Workshop

1. At the system prompt, type:
   % /usr/sadm/bin/cimworkshop
   The CIM Workshop Login dialog box displays.

2. Follow the instructions in the context-sensitive help to fill in the fields in the Login dialog box. Then click OK.
   The CIM Workshop main window displays.

▼ How to Exit CIM Workshop

- From the CIM Workshop main window, choose Workshop->Exit.
  CIM Workshop exits.

About the Main Window

The main window in CIM Workshop is divided into three panes:

- **Left pane** – Displays the class inheritance tree of the current name space.
- **Right pane** – Displays the Properties, Methods, and Events tabs. When you select a class in the left pane, click one of these tabs in the right pane to display more information on the properties, methods, or events of the selected class.
- **Bottom pane** – Displays notification when events occur to which you are subscribed.
CHAPTER 2

Using the CIM Object Manager

The Common Information Model (CIM) Object Manager is software that transfers CIM data between WBEM client applications and managed resources.

This chapter discusses the following topics:

- “About the CIM Object Manager” on page 31
- “init.wbem Command” on page 32
- “Stopping and Restarting the CIM Object Manager” on page 33
- “Exception Messages” on page 35

About the CIM Object Manager

The CIM Object Manager (CIMOM) manages CIM objects on a WBEM-enabled system. A CIM object is a representation, or model, of a managed resource, such as a printer, disk drive, or CPU. CIM objects are stored internally as classes that are written in the Java programming language.

When a WBEM client application accesses information about an object, the CIMOM contacts either the provider for that object or the CIM Object Manager Repository. Providers are classes that communicate with managed objects to access data. A WBEM client application might request data from a managed resource that is not available from the CIM Object Manager Repository. In this case, the CIM Object Manager forwards the request to the provider for that managed resource. The provider dynamically retrieves the information.

At startup, the CIM Object Manager performs the following functions:

- Listens for RMI connections on port 5987 and for XML over HTTP connections on port 5988
- Sets up a connection to the CIM Object Manager Repository
- Waits for incoming requests

The CIM Object Manager:
**init.wbem Command**

The `init.wbem` command is automatically run during installation, and then each time you reboot a system. The `init.wbem` command starts the CIM Object Manager and Solaris Management Console server, both of which run combined in a single process. You can also use `init.wbem` to stop the CIM Object Manager, to stop the Solaris Management Console server, or to retrieve status from a server. You can find additional information about this command in the `init.wbem(1M)` man page.

Generally, you do not need to stop the CIM Object Manager. However, if you change an existing provider, you must stop and restart the CIM Object Manager before using the updated provider.

You can specify three options with `init.wbem`:

- **start** – Starts the CIM Object Manager or Solaris Management Console server on the local host
- **stop** – Stops the CIM Object Manager and Solaris Management Console server on the local host
- **status** – Gets status for the CIM Object Manager and Solaris Management Console server on the local host

**Solaris Management Console Server**

The Solaris Management Console software provides Solaris platform management applications such as User Manager, Disk Manager, and Log Viewer. The Solaris Management Console server provides tools that the console can download. The server also performs common services for the console and its tools, such as authentication, authorization, logging, messaging, and persistence.
The Solaris Management Console is described in other chapters in this document. For more information see the *System Administration Guide: Basic Administration*.

**System Booting**

The `init.wbem` command is located in the `/etc/init.d` directory. The file `/etc/rc2.d/S90wbem` runs with the `start` option when initialization state 2 is entered, normally at boot time. The files `/etc/rc0.d/K36wbem`, `/etc/rc1.d/K36wbem`, and `/etc/rcS.d/K36wbem` are run with the `stop` option when initialization states 0, 1, and 5 are entered.

**Stopping and Restarting the CIM Object Manager**

If you change a provider, you must stop and restart the CIM Object Manager before using the updated provider.

**How to Stop the CIM Object Manager**

1. Become superuser.
2. Stop the CIM Object Manager.
   
   ```
   # /etc/init.d/init.wbem stop
   ```

**How to Restart the CIM Object Manager**

1. Become superuser.
2. Restart the CIM Object Manager.
   
   ```
   # /etc/init.d/init.wbem start
   ```

**Upgrading the CIM Object Manager Repository**

If you have upgraded to the Solaris 10 platform from a previous version of the Solaris software, you must update any proprietary custom Managed Object Format (MOF) data to the new repository format that was introduced in Solaris 9. Any CIM and Solaris platform MOF data that you modified prior to the upgrade is destroyed by the upgrade. As a result, you will either need to recompile your MOF files or merge the WBEM data after you upgrade.
Caution – Failure to recompile or merge the modified data results in data loss.

Use the following table to determine whether to recompile or merge the WBEM data after you upgrade to the Solaris 10 OS.

<table>
<thead>
<tr>
<th>Environment Before Upgrade</th>
<th>Recompile Proprietary Managed Object Format (MOF) Files?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris 8 (Solaris WBEM Services 2.0)</td>
<td>Yes</td>
</tr>
<tr>
<td>Solaris 8 6/00 (WBEM Services 2.0)</td>
<td></td>
</tr>
<tr>
<td>Solaris 8 10/00 (WBEM Services 2.2)</td>
<td></td>
</tr>
<tr>
<td>Solaris 8 1/01 (WBEM Services 2.3)</td>
<td>No. However, you need to merge the data into an upgraded repository.</td>
</tr>
<tr>
<td>Solaris 8 4/01 (WBEM Services 2.4)</td>
<td></td>
</tr>
<tr>
<td>Solaris 8 7/01 (WBEM Services 2.4)</td>
<td></td>
</tr>
<tr>
<td>Solaris 8 10/01 (WBEM Services 2.4)</td>
<td></td>
</tr>
<tr>
<td>Solaris 9 Any release (WBEM Services 2.5)</td>
<td></td>
</tr>
</tbody>
</table>

How to Recompile Your MOF Files

1. Upgrade your system to the Solaris 10 OS.
2. Become superuser.
3. Change directory to the location of your proprietary MOF files.
4. Use the mofcomp command to compile each of your proprietary MOF files.
   
   ```bash
   # /usr/sadm/bin/mofcomp root root-passwd MOF-filename
   ```

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

5. Stop the CIM Object Manager.
   
   ```bash
   # /etc/init.d/init.wbem stop
   ```

6. Start the CIM Object Manager.

   ```bash
   # /etc/init.d/init.wbem start
   ```
The CIMOM adds repository files that contain the converted data to the directory `/var/sadm/wbem/logr/`. This directory was created when you upgraded your system to the Solaris 10 OS.

## How to Merge WBEM Data

1. **Upgrade your system to the Solaris 10 OS.**

2. **Become superuser.**

3. **Stop the CIM Object Manager.**
   ```
   # /etc/init.d/init.wbem stop
   ```

   **Caution** – Failure to stop the CIM Object Manager before you run `wbemconfig convert` might corrupt your data.

4. **Merge the original data in the previous Reliable Log with the data in the Solaris 10 Reliable Log.**
   ```
   # /usr/sadm/lib/wbem/wbemconfig convert
   ```

   **Note** – The `wbemconfig convert` command successfully converts any proprietary custom MOF data, but not any CIM or Solaris MOF data that you have modified. CIM and Solaris platform MOF data that you have modified is destroyed. To recompile any modified CIM or Solaris platform MOF data in the new repository, use the `mofcomp` command to compile the MOF files that contain the class definitions.

### Exception Messages

The CIM Object Manager generates exception messages to indicate incorrect MOF syntax and semantics. Chapter 9, “Troubleshooting,” contains information about exception messages.
This chapter describes the sample programs provided with the Solaris WBEM SDK, and includes the following topics:

- “About the Sample Programs” on page 37
- “Sample Applet” on page 38
- “Sample Client Programs” on page 38
- “Sample Provider Programs” on page 40

**About the Sample Programs**

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

**Note** – To use the applet and sample programs, make sure that /usr/java points to at least JDK 1.2.2, and that the program files are installed in the /usr/demo/wbem directory.

The following samples are provided:

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

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

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

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

For more information on the JDK appletviewer or the Java runtime environment, see http://java.sun.com.

▼ How to Run the Sample Applet Using Appletviewer

To run the sample applet using appletviewer, type the following command:

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

▼ How to Run the Sample Applet in a Web Browser

To run the sample applet in a web browser, open the following file in your web browser:

```
```

Sample Client Programs

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>Program</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>./batching</td>
<td>rmi</td>
<td>http ./TestBatch host username password classname</td>
</tr>
</tbody>
</table>
Running the Sample Client Programs

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

- Set the CLASSPATH environment variable using one of the following methods:
  - To use the C shell, type:
    ```bash
    % setenv CLASSPATH .:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar:/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
    ```
  - To use the Bourne shell, type:
    ```bash
    % setenv CLASSPATH .:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar:/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
    ```

**How to Run the Sample Client Programs**

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

- Run the sample client programs using the following format:
  ```bash
  % java program_name parameters
  ```

**Example 3–1 Running a Sample Client Program**

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

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

**Sample Provider Programs**

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

**Table 3-2: Sample Provider Programs**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeProvider.java</td>
<td>Top-level provider program that fulfills requests from the CIM Object Manager and routes these requests to the Native_Example provider. This program implements the instanceProvider and methodProvider APIs. This program also declares methods to enumerate instances and to get an instance of the Native_Example class. This program also declares a method that invokes a method to print the string “Hello World.”</td>
</tr>
</tbody>
</table>
TABLE 3-2  Sample Provider Programs  (Continued)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native_Example.mof</td>
<td>Creates a class that registers the NativeProvider provider with the CIM Object Manager. This MOF file identifies NativeProvider as the provider to service requests for dynamic data in the Native_Example class. The file also declares the properties and methods to be implemented by the NativeProvider.</td>
</tr>
<tr>
<td>Native_Example.java</td>
<td>The NativeProvider program calls this provider to enumerate instances and to get an instance of the Native_Example class. The Native_Example provider uses the APIs to enumerate objects and to create instances of objects. The Native_Example class declares native methods, which call C functions in the native.c file to get system-specific values. Examples of such values include host name, serial number, release, machine, architecture, and manufacturer.</td>
</tr>
<tr>
<td>native.c</td>
<td>This C program implements methods in the Native_Example Java provider in native C code.</td>
</tr>
<tr>
<td>Native_Example.h</td>
<td>A machine-generated header file for the Native_Example class. Defines the correspondence between the names of native method in the Java programming language and the native C functions that execute those methods.</td>
</tr>
<tr>
<td>libnative.so</td>
<td>Binary native C code compiled from the native.c file.</td>
</tr>
</tbody>
</table>

▼ How to Run the Sample Provider Programs

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

1. Set the LD_LIBRARY_PATH environment variable to the location of the provider class files.
   - To use the C shell, type:
     ```bash
     % setenv LD_LIBRARY_PATH /usr/sadm/lib/wbem
     ```
   - To use the Bourne shell, type:
     ```bash
     % LD_LIBRARY_PATH = /usr/sadm/lib/wbem
     ```

2. Copy the libnative.so shared library file to the directory specified by the LD_LIBRARY_PATH environment variable.
   ```bash
   % cp libnative.so /usr/sadm/lib/wbem
   ```

3. Move the provider class files to the same path as the package to which those files belong.
   ```bash
   % mv *.class /usr/sadm/lib/wbem
   ```

Stop the CIM Object Manager in the same shell in which you set the LD_LIBRARY_PATH environment variable.

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

**Note** – When you set the LD_LIBRARY_PATH environment variable in a shell, you must stop and restart the CIMOM in the same shell to recognize the new variable.

Start the CIM Object Manager.

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

Exit being superuser.

Compile the program’s associated .mof file to load the appropriate class in the CIMOM and to identify the provider.

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

Start CIM Workshop.

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

In the CIM Workshop toolbar, click the Find Class icon.

In the Input dialog box, type the name of the class that you want to display and then click OK. The class displays in CIM Workshop.
Writing a Client Program

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

- “Client API Overview” on page 43
- “Opening and Closing a Client Connection” on page 44
- “Performing Basic Client Operations” on page 46
- “Setting Access Control” on page 65
- “Working With Qualifiers and Qualifier Types” on page 68
- “Batching Client Requests” on page 69
- “Handling CIM Events” on page 71
- “Reading and Writing Log Messages” on page 79

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

Client API Overview

WBEM client applications use the javax.wbem.client APIs to manipulate Common Information Model (CIM) objects. A client application uses the CIM API to construct an object and then to create an instance of that object. Examples of such an object include a class, an instance, or a name space. The application uses the client APIs to pass the object to the CIM Object Manager and request a WBEM operation. Examples of such an operation are creating a CIM class, creating an instance, or creating a name space.

Sequence of a Client Application

Client applications typically follow this sequence:
1. Connect to the CIMOM using CIMClient. A client application connects to the CIMOM each time the client needs to perform a WBEM operation. These operations might include creating a CIM class and updating a CIM instance. See “Opening and Closing a Client Connection” on page 44.

2. Use the client APIs to request operations and to perform programming tasks. The application’s feature set determines which operations it needs to request. The tasks that most programs perform include:
   ■ Create, delete and update instances
   ■ Enumerate objects
   ■ Call methods
   ■ Retrieve class definitions
   ■ Handle errors

   Client programs can also create and delete classes, create and delete name spaces, and use qualifiers. See “Performing Basic Client Operations” on page 46.

3. Close the client connection to the CIM Object Manager using CIMClient, to free the server resources used by the client session. See “Opening and Closing a Client Connection” on page 44.

Opening and Closing a Client Connection

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

About Name Spaces

When an application connects to the CIMOM, the application must also connect to a name space, where all subsequent operations occur. A name space is a directory-like structure that contains classes, instances, and qualifier types. The names of all objects within a name space must be unique. When you install the Solaris WBEM SDK, four name spaces are created:
   ■ root\cimv2 – The default name space. Contains the CIM classes that represent objects on the system on which Solaris WBEM software is installed.
   ■ root\security – Contains the security classes.
   ■ root\snmp – Contains the SNMP adapter classes.
   ■ root\system – Contains the classes that manage the CIM Object Manager.
Opening a Client Connection

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

- **name** – Required. An instance of a CIMNameSpace object that contains the name of the host and the name space used for the client connection. The default value is root\cimv2 on the local host. The local host is the same host in which the client application is running. Once the client is connected to the CIMOM, all subsequent CIMClient operations occur within the specified name space.

- **principal** – Required. An instance of a UserPrincipal object that contains the name of a valid Solaris OS user account. The CIMOM checks the access privileges for the user name to determine the type of access that is allowed to CIM objects.

- **credential** – Required. An instance of a PasswordCredential object that contains a valid password for the UserPrincipal Solaris OS account.

- **protocol** – Optional (string). Protocol that is used for sending messages to the CIMOM. Possible values are RMI, which is the default value, or HTTP.

**EXAMPLE 4-1  Connecting to the Root Account**

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

```java
...
/* Create a name space object initialized with two null strings
that specify the default host (the local host) and the default
name space (root\cimv2). */
CIMNameSpace cns = new CIMNameSpace("", "");
UserPrincipal up = new UserPrincipal("root");
PasswordCredential pc = new PasswordCredential("root-password");
/* Connect to the name space as root with the root password. */
CIMClient cc = new CIMClient(cns, up, pc);
...
```

**EXAMPLE 4-2  Connecting to a User Account**

In this example, the application first creates an instance of a CIMNameSpace, UserPrincipal, and PasswordCredential object. Then, the application uses the CIMClient class to pass the host name, name space, user name, and password credential in order to create a connection to the CIMOM.

```java
...
```
/* Create a name space object initialized with A (name of name space) on host happy. */
CIMNameSpace cns = new CIMNameSpace("happy", "A");
UserPrincipal up = new UserPrincipal("Mary");
PasswordCredential pc = new PasswordCredential("marys-password");
CIMClient cc = new CIMClient(cns, up, pc);
...

EXAMPLE 4–3  Authenticating as an RBAC Role Identity
You use the SolarisUserPrincipal and SolarisPasswordCredential classes to authenticate a user’s role identity. This example authenticates as Mary and assumes the role Admin.

  {
    CIMNameSpaceRole cns = new CIMNameSpace("happy", "A");
    SolarisUserPrincipal sup = new SolarisUserRolePrincipal("Mary", "Admin");
    SolarisPswdCredential spc = new SolarisPswdCredential("marys-password", "admins-password");
    CIMClient cc = new CIMClient(cns, sup, spc);
  }

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

EXAMPLE 4–4  Closing a Client Connection
This example closes a client connection. The instance variable cc represents the client connection.

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

Performing Basic Client Operations
This section describes how to use the javax.wbem.client APIs to request operations and to perform common programming tasks.

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

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

**EXAMPLE 4-5  Creating an Instance**

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

```java
...{
    /*Connect to the CIM Object Manager in the root\cimv2 name space on the local host. Specify the username and password of an account that has write permission to the objects in the root\cimv2 name space. */
    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root-password");
    /* Connect to the name space as root with the root password. */
    CIMClient cc = new CIMClient(cns, up, pc);
    ...;

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

    /* Create a new instance of the Solaris Package class populated with the default values for properties. If the provider for the class does not specify default values, the values of the properties will be null and must be explicitly set. */
    CIMInstance ci = cc.CreateInstance (new CIMObjectPath("Solaris_Package"),
                                        ci);
}
...;
```

**Deleting an Instance**

Use the `deleteInstance` method to delete an instance.

**EXAMPLE 4-6  Deleting Instances**

The example does the following:

- Connects the client application to the CIMOM
- Uses `CIMObjectPath` to construct an object containing the CIM object path of the object to be deleted
- Calls `enumerateInstance` to get the specified instance and all instances of its subclasses
EXAMPLE 4–6  Deleting Instances  (Continued)

- Calls deleteInstance to delete each instance

```java
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]) name space (args[3] and classname (args[4])
 * It will delete all instances of the specified classname. The specified
 * username must have write permissions to the specified name space
 */
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
            // name space. We create a CIMNameSpace (cns) pointing to
            // the specified name space on the specified host
            CIMNameSpace cns = new CIMNameSpace(args[0], args[3]);
            // 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]);

            // 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
```
EXAMPLE 4–6  Deleting Instances    (Continued)

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

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

Getting and Setting Instances

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

- If localOnly is true, getInstance returns only the noninherited properties in the specified
  instance. The noninherited properties are defined in the instance.
- If localOnly is false, all properties in the class are returned. These properties include those
  properties that are defined in the instance, and all properties inherited from all parent
  classes in its class hierarchy.

Use the setInstance method to update an existing instance.

EXAMPLE 4–7  Getting and Setting Instances

This example does the following:

- Gets instances of an object path in an enumeration
- Updates the property value of b to 10 in each instance
- Passes the updated instances to the CIMOM

...
EXAMPLE 4–7  Getting and Setting Instances  (Continued)

    CIMObjectPath cop = new CIMObjectPath("myclass");

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

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

...

Getting and Setting Properties

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

EXAMPLE 4–8  Getting a Property

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

    ...
    {  /* Create an object (CIMObjectPath) to store the name of the Solaris_Processor class. */
        CIMObjectPath cop = new CIMObjectPath("Solaris_Processor");

        /* The CIM Object Manager returns an enumeration containing the names of instances of the Solaris_Processor class. */
        Enumeration e = cc.enumerateInstanceNames(cop);

        /* Iterate through the enumeration of instance object paths. Use the getProperty method to get the current clockspeed value for each Solaris processor. */
        while(e.hasMoreElements()) {
            CIMValue cv = cc.getProperty(e.nextElement(CIMObjectPath),
                "CurrentClockSpeed");
            System.out.println(cv);
        }
    }
    ...
EXAMPLE 4–9  Setting a Property

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

```java
...
{
   /* Create an object (CIMObjectPath) to store the name of the Solaris_Processor class. */
   CIMObjectPath cop = new CIMObjectPath("Solaris_UserTemplate");

   /* The CIM Object Manager returns an enumeration containing the names of instances of the Solaris_UserTemplate class and all its subclasses. */
   Enumeration e = cc.enumerateInstanceNames(cop);

   /* Iterate through the enumeration of instance object paths. Use the setProperty method to set the initial shell value to /usr/bin/sh for each Solaris_UserTemplate instance. */
   for (; e.hasMoreElements(); cc.setProperty(e.nextElement(), 
      "/usr/bin/sh", new CIMValue(new Integer(500))));
}
```

Enumerating Objects

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

**Enumerating Objects**

<table>
<thead>
<tr>
<th>Method</th>
<th>No args</th>
<th>deep</th>
<th>localOnly</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumerateClasses</td>
<td>Returns the contents of the class specified in path.</td>
<td>If true: Returns the contents of the subclasses of the specified class, but does not return the class. If false: Returns the contents of the direct subclasses of the specified class.</td>
<td>If true: Returns only noninherited properties and methods of the specified class. If false: Returns all properties of the specified class.</td>
</tr>
</tbody>
</table>
### Enumerating Objects (Continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>deep</th>
<th>localOnly</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumerateInstances</td>
<td>Returns the instances of the class specified in path.</td>
<td>If true: Returns the instances of the specified class and its subclasses.</td>
<td>If true: Returns only noninherited properties of the instances of the specified class.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If false: Returns the instances of the specified class and its subclasses. The properties of the subclasses are filtered out.</td>
<td>If false: Returns all properties of the instances of the specified class.</td>
</tr>
<tr>
<td>enumerateClassNames</td>
<td>Returns the names of the class specified in path.</td>
<td>If true: Returns the names of all classes derived from the specified class.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If false: Returns only the names of the first-level children of the specified class.</td>
<td>N/A</td>
</tr>
<tr>
<td>enumerateInstanceNames</td>
<td>Returns the names of the instances of the class specified in path.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>enumNameSpace</td>
<td>Returns a list of the name spaces within the name space specified in path.</td>
<td>If true: Returns the entire hierarchy of name spaces under the specified name space.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If false: Returns only the first level children of the specified name space.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### EXAMPLE 4–10 Enumerating Classes

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

```java
...
{
  /* 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 noninherited methods and properties for each class and subclass (localOnly is true). */
  Enumeration e = cc.enumerateClasses(cop, true, true);
}
...```
The following example program performs a deep and shallow (deep=false) enumeration of classes and instances. The localOnly flag returns the contents of the classes and instances instead of the names of the classes and instances.

```java
import java.rmi.*;
import java.util.Enumeration;
import javax.wbem.client.CIMClient;
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;

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

    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        CIMObjectPath cop = null;
        if (args.length < 4) {
            System.out.println("Usage: ClientEnum host user passwd " + "classname");
            System.exit(1);
        }
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            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
```
EXAMPLE 4–11  Enumerating Classes and Instances  (Continued)

```java
e = cc.enumerateInstances(cop, false, true, true, true, null);
// Will print out all the instances of the class and its
// subclasses.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
// Do a shallow enumeration of the instances of the class
e = cc.enumerateInstances(cop, false, false, true, true, null);
// Will print out all the instances of the class.
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
e = cc.enumerateInstanceNames(cop);
while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
System.out.println("+++++");
e = cc.enumerateInstanceNames(cop);
while (e.hasMoreMoreElements()) {
    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();
}
```

EXAMPLE 4–12  Enumerating Class Names

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

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

    /* This enumeration contains the names of the classes and subclasses 
    in the enumerated class. */
    Enumeration e = cc.enumerateClassNames(cop, true);
}
...
EXAMPLE 4–13  Enumerating Name Spaces

This example program uses the enumNameSpace method in the CIMClient class to print the name of the name space and all the name spaces contained within the name space.

```java
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");
 *       System.exit(1);
 *     }
 *     try {
 *       // args[0] contains the hostname. We create a CIMNameSpace
 *       // (cns) pointing to the specified name space on the
 *       // specified host
 *       CIMNameSpace cns = new CIMNameSpace(args[0], "");
 *       // 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 name space (args[3]) to create a CIMObjectPath
 *       CIMObjectPath cop = new CIMObjectPath("", args[3]);
 *       // Enumerate the name space
 *       Enumeration e = cc.enumNameSpace(cop);
 *       while (e.hasMoreElements()) {
 *         System.out.println((CIMObjectPath)e.nextElement());
 *       } // end while
 *     }
 *   }
 * }
```

Performing Basic Client Operations

Chapter 4 • Writing a Client Program 55
Creating Associations

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

The preceding figure shows two classes, Teacher and Student. Both classes are linked by the TeacherStudent association. The TeacherStudent association has two references:

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

About the Association Methods

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>associators</td>
<td>Gets the CIM classes or instances that are associated with the specified CIM class or instance</td>
</tr>
</tbody>
</table>
These methods take a single required argument, CIMObjectPath. CIMObjectPath is the name of a source CIM class or CIM instance whose associations, associated classes, or instances you want to return. If the CIMOM does not find any associations, associated classes, or instances, the CIMOM does not return anything.

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

In the preceding figure, the associators and associatorNames methods return information about the classes associated with the Teacher and Student classes. The references and referenceNames methods return information about the associations between the Teacher and Student classes.
### TABLE 4–3 TeacherStudent Methods

<table>
<thead>
<tr>
<th>Example</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>associators(Teacher, null, null, null, null, false, false, null)</td>
<td>Student class</td>
<td>Returns associated classes. Student is linked to Teacher by the TeacherStudent association.</td>
</tr>
<tr>
<td>associators(MathTeacher, null, null, null, null, false, false, null)</td>
<td>Student</td>
<td>Returns associated classes. Teacher is linked to Student by the TeacherStudent association. MathTeacher and ArtTeacher inherit the TeacherStudent association from Teacher.</td>
</tr>
<tr>
<td>associatorNames(Teacher, null, null, null)</td>
<td>Name of the Student class</td>
<td>Returns the names of the associated classes. Student is linked to Teacher by the TeacherStudent association.</td>
</tr>
<tr>
<td>references(Student, null, null, false, false, null)</td>
<td>TeacherStudent</td>
<td>Returns the associations in which Student participates.</td>
</tr>
<tr>
<td>references(Teacher, null, null, false, false, null)</td>
<td>TeacherStudent</td>
<td>Returns the associations in which Teacher participates.</td>
</tr>
<tr>
<td>referenceNames(Teacher, null, null)</td>
<td>Name of the TeacherStudent class</td>
<td>Returns the names of the associations in which Teacher participates.</td>
</tr>
<tr>
<td>referenceNames(Teacher, null, null)</td>
<td>Name of the TeacherStudent class</td>
<td>Returns the names of the associations in which Teacher participates.</td>
</tr>
</tbody>
</table>

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

### Passing a Class to the Association Methods

To specify the name of a class, you specify its *model path*. The model path includes the class’s name space, class name, and keys. A *key* is a property or set of properties that uniquely identify managed resource. Key properties are marked with the key qualifier. The following example shows a sample model path:

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

This model path specifies the following values:
- `\myserver\root\cimv2` is the default CIM name space on host `myserver`. 
Solaris_ComputerSystem is the name of the class from which the instance is derived.

Name=mycomputer, CreationClassName=Solaris_ComputerSystem are two key properties in the format key property=value.

Passing Instances to the Association Methods

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

EXAMPLE 4–14  Passing Instances

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

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

Using Optional Arguments With the Association Methods

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

You can pass values for any one or a combination of the optional parameters. You must enter a value or null for each parameter. The first four parameters are used to filter the classes and instances that are returned:

- assocClass
- resultClass
- resultRole
- role

Only the classes and instances that match the values specified for these parameters are returned. The includeQualifiers, includeClassOrigin, and propertyList parameters filter the information that is included in the classes and instances that are returned.
Calling Methods

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

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

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

EXAMPLE 4–15  Calling a Method

This example gets the instances of the `CIM_Service` class, which represent services that manage device or software features. The example uses the `invokeMethod` method to stop each service.

```java
{
    /* Pass the CIM Object Path of the CIM_Service class
    to the CIM Object Manager. We want to invoke a method defined in
    this class. */
    CIMObjectPath op = new CIMObjectPath("CIM_Service");

    /* The CIM Object Manager returns an enumeration of instance
    object paths, the names of instances of the CIM_Service
    class. */
    Enumeration e = cc.enumerateInstanceNames (op, true);

    /* Iterate through the enumeration of instance object paths */
    while(e.hasMoreElements()) {
        // Get the instance
        CIMObjectPath op = (CIMObjectPath) e.nextElement();
        //Invoke the Stop Service method to stop the CIM services.
        cc.invokeMethod("StopService", null, null);
    }
}
Retrieving Class Definitions

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

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

**EXAMPLE 4-16  Retrieving a Class Definition**

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

- `CIMNameSpace` – Create a new name space
- `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

```java
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
 * name space root\cimv2.
 */
public class GetClass {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal("root");
            PasswordCredential pc = new PasswordCredential("root_password");
            cc = new CIMClient(cns);
            CIMObjectPath cop = new CIMObjectPath(args[1]);
            // Returns only the methods and properties that
            // are local to the specified class (localOnly is true).
            cc.getClass(cop, true);
        } catch (Exception e) {
            System.out.println("Exception: "+e);
        }
        if(cc != null) {
            cc.close();
        }
    }
}
```
Handling Exceptions

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

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

Creating a Name Space

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

EXAMPLE 4–17 Creating a Name Space

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

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

```java
{ ... /* Creates a name space 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 name space (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 name space object (cns), which contains the host name (args[0]) and parent name space name (args[1]), a user name string (args[3]), and a password string (args[4]). */
```
Creating a Name Space (Continued)

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

/* Passes to the CIM Object Manager another name space object that contains a null string (host name) and args[2], the name of a child name space (for example, secondlevel). */
CIMNameSpace cop = new CIMNameSpace("", args[2]);

/* Creates a new name space by the name passed in as args[2] under the toplevel name space on myhost. */
cc.createNameSpace(cop);
```

Deleting a Name Space

Use the `deleteNameSpace` method to delete a name space.

Creating a Base Class

**Note** – You can also create a base class using the MOF language. If you are familiar with MOF syntax, use a text editor to create a MOF file. Then use the MOF compiler to compile the file into Java classes. See Chapter 7, “Creating JavaBeans Components Using the MOF Compiler.”

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

A property can have a key qualifier, which identifies the property 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 name space, you must first compile the core MOF files into the name space. The core MOF files contain the declarations of the standard CIM qualifiers, such as the *key* qualifier.

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

Use the CIMClient method, deleteClass, to delete a class. This method removes the class and throws a CIMException.

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

**EXAMPLE 4–18 Deleting a Class**

This example uses the deleteClass method to delete a class in the default name space root\cimv2. This program takes four required string arguments:

- hostname
- classname
- username
- password

The user running this program must specify the username and the password for an account, which has write permission to the root\cimv2 name space.

```java
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
 * name space 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
            // pointing to the default name space on the specified host
            CIMNameSpace cns = new CIMNameSpace(args[0]);

            // We create a UserPrincipal (up) using the username and
            // a PasswordCredential using the password.
```
Setting Access Control

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

- **Solaris_Acl** – Base class for Solaris OS access control lists (ACLs). This class defines the
  string property *capability* and sets its default value to *r* (read only).
- **Solaris_UserAcl** – Represents a user's access control to the CIM objects within the
  specified name space.
- **Solaris_NameSpaceAcl** – Represents the access control on a name space.

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

Combine the use of these two classes by using the Solaris_NameSpaceACL class to first restrict
access for all users to the objects in a name space. Then, you can use the Solaris_UserACL class
to grant selected users access to the name space.

**Solaris_UserAcl Class**

The Solaris_UserAcl class extends the Solaris_Acl base class, from which it inherits the
string property *capability* with a default value of *r* (read only). You can set the *capability*
property to any one of the values for access privileges shown in the following table.
<table>
<thead>
<tr>
<th>Access Right</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>Read</td>
</tr>
<tr>
<td>( rw )</td>
<td>Read and Write</td>
</tr>
<tr>
<td>( w )</td>
<td>Write</td>
</tr>
<tr>
<td>none</td>
<td>No access</td>
</tr>
</tbody>
</table>

The `Solaris_UserAcl` class defines the key properties that are shown in the following table. Only one instance of the name space and user name ACL pair can exist in a name space.

<table>
<thead>
<tr>
<th>Property</th>
<th>DataType</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>nspace</td>
<td>string</td>
<td>Identifies the name space to which the ACL applies</td>
</tr>
<tr>
<td>username</td>
<td>string</td>
<td>Identifies the user to which the ACL applies</td>
</tr>
</tbody>
</table>

### To Set Access Control for a User

1. **Create an instance of the `Solaris_UserAcl` class.**
   
   ```java
   /* Create a name space object initialized with root\security (name of name space) on the local host. */
   CIMNameSpace cns = new CIMNameSpace("", "root\security");
   
   // Connect to the root\security name space as root.
   cc = new CIMClient(cns, user, user_passwd);
   
   // Get the `Solaris_UserAcl` class
   cimclass = cc.getClass(new CIMObjectPath("Solaris_UserAcl");
   
   // Create a new instance of the `Solaris_UserAcl` class
   ci = cimclass.newInstance();
   ```

2. **Set the `capability` property to the desired access rights.**
   
   ```java
   /* Change the access rights (capability) to read/write for user Guest on objects in the root\molly name space. */
   ci.setProperty("capability", new CIMValue(new String("rw")));
   ci.setProperty("nspace", new CIMValue(new String("root\molly")));
   ci.setProperty("username", new CIMValue(new String("guest")));
   ```
3 Update the instance.

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

### Solaris\_NamespaceAcl Class

The Solaris\_NamespaceAcl class extends the Solaris\_Acl base class and inherits the string property `capability` with a default value `r` (read-only for all users). The Solaris\_NamespaceAcl class defines this key property:

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>nspace</td>
<td>string</td>
<td>Identifies the name space to which the access control list applies. Only one instance of the name space ACL can exist in a name space.</td>
</tr>
</tbody>
</table>

#### To Set Access Control for a Name Space

1 Create an instance of the Solaris\_namespaceAcl class.

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

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

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

// Create a new instance of the Solaris\_namespaceAcl class  
class ci = cimclass.newInstance();

...  

2 Set the `capability` property to the desired access rights.

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

...  

3 Update the instance.

// Pass the updated instance to the CIM Object Manager  
cc.createInstance(new CIMObjectPath(), ci);
Working With Qualifiers and Qualifier Types

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

- Type
- Value
- Name

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

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

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

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

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

Getting and Setting CIM Qualifiers

A qualifier flavor is a flag that governs the use of a qualifier. Flavors describe rules that specify whether a qualifier can be propagated to derived classes and instances. Rules also determine whether a derived class or instance can override the qualifier’s original value.

**EXAMPLE 4–19  Setting CIM Qualifiers**

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

```
{
  try {
    cimSuperClass = cimClient.getClass(new CIMObjectPath(scName));
    Vector v = new Vector();
    for (Enumeration e = cimSuperClass.getQualifiers().elements();
      e.hasMoreElements()) {
      CIMQualifier qual = (CIMQualifier)
EXAMPLE 4–19 Setting CIM Qualifiers (Continued)

```java
((CIMQualifier)e.nextElement()).clone();
    v.addElement(qual);
} catch (CIMException exc) {
    cimClass.setQualifiers(v);
} return;
```

Batching Client Requests

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

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

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

The client executes the batch operation list by invoking the performBatchOperations method of CIMClient. The results of the batch operation are returned in a BatchResult object. Clients can then pass the operation ID to the getResult method of the BatchResult class to get the results of the operations. If an operation on the list generates an exception, an exception object is embedded in the BatchResult object. When you invoke the getResult method with the ID of the operation that failed, the exception is thrown by the getResult method.

EXAMPLE 4–20 Batching Example

The following example shows how you can use the batching API to perform multiple operations in one remote call. In this example, three operations are performed as a single batch operation. The operations are enumerateInstanceNames, getClass, and enumerateInstances.
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Vector;
import java.lang.String;
import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;

public class TestBatch {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        CIMObjectPath cop = null;
        String protocol = CIMClient.CIM_RMI;
        if (args.length < 4) {
            System.out.println("Usage: TestBatch host user passwd
classname " + [rmi|http]);
            System.exit(1);
        }
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal(args[1]);
            PasswordCredential pc = new PasswordCredential(args[2]);
            if (args.length == 5 && args[4].equalsIgnoreCase("http")) {
                protocol = CIMClient.CIM_XML;
            }
            cc = new CIMClient(cns, up, pc, protocol);
            CIMObjectPath op = new CIMObjectPath(args[3]);
            BatchCIMClient bc = new BatchCIMClient();
            int[] ids = new int[3];
            ids[0] = bc.enumerateInstanceNames(op);
            ids[1] = bc.getClass(op, false, true, true, null);
            ids[2] = bc.enumerateInstances(op, true, false, false,
                                      false, null);
            BatchResult br = cc.performBatchOperations(bc);
            Enumeration instanceNames = (Enumeration)br.getResult
                                      (ids[0]);
            CIMClass cl = (CIMClass)br.getResult(ids[1]);
            Enumeration instances = (Enumeration)br.getResult(ids[2]);
            while (instanceNames.hasMoreElements()) {
                System.out.println((CIMObjectPath)instanceNames.
                                      nextElement());
            }
            System.out.println(cl.toMOF());
            while (instances.hasMoreElements()) {
                System.out.println((CIMObjectPath)instances.
                                      nextElement());
            }
        }
    }
}
EXAMPLE 4–20  Batching Example  (Continued)

```java
System.out.println((CIMInstance)instances.
    nextElement());
}
}
catch (Exception e) {
    e.printStackTrace();
    System.out.println("Exception: "+e);
}
// close session.
if (cc != null) {
    cc.close();
}
```

Handling CIM Events

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

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

An indication may have zero or more *triggers*, which are recognitions of changes in state. WBEM does no have an explicit object representing a trigger. Instead, a trigger is implied by the following actions:

- An operation on a basic object of a system. An operation could create, delete, modify, or access a class. An operation might also modify or access an instance.

- Any event that takes place in the managed environment.

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

You can view the related CIM event classes in the Solaris WBEM Services schema at file:/usr/sadm/lib/wbem/doc/mofhtml/index.html. The class is structured as shown in the following table.
### Table 4–5  CIM_Indication Class Structure

<table>
<thead>
<tr>
<th>Root Class</th>
<th>Superclass</th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIM_Indication</td>
<td>CIM_ClassIndication</td>
<td>CIM_ClassCreation, CIM_ClassDeletion, CIM_ClassModification</td>
</tr>
<tr>
<td>CIM_InstIndication</td>
<td>CIM_InstCreation</td>
<td>CIM_InstDeletion, CIM_InstMethodCall, CIM_InstModification, CIM_InstRead</td>
</tr>
<tr>
<td>CIM_ProcessIndication</td>
<td>CIM_AlertIndication</td>
<td>CIM_AlertIndication, CIM_AlertInstIndication, CIM_ThresholdIndication, CIM_SNMPTrapIndication</td>
</tr>
</tbody>
</table>

### About Indications

CIM events can be classified as either life cycle or process. A *life cycle event* is a built-in CIM event that occurs in response to a change to a particular change in data. The types of changes that trigger a life cycle event are:

- A class is created, modified, or deleted
- A class instance is created, modified, deleted, read, or has a method invocation

A *process event* is a user-defined event that is not described by a life cycle event.

Event providers generate indications in response to requests made by the CIMOM. The CIMOM analyzes subscription requests. The CIMOM uses the EventProvider or the CIM_IndicationProvider interface to contact the provider, requesting the provider to generate the appropriate indications. When the provider generates the indication, the CIMOM 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. There is a sequence of steps that the CIMOM follows in the case of subscriptions pertaining to instance life cycle indication, such as subclasses of CIM_InstIndication. Once the CIMOM determines the classes covered by the subscription, the CIMOM contacts the instance providers for those classes. For process indications, the CIMOM contacts the appropriate provider using the Provider qualifier.

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

- The CIMOM handles the following events, if either the provider does not support indications or the provider tells the CIMOM not to poll:
The CIM Object Manager Repository handles all class indications and life cycle indications for classes that do not have providers. These classes include the following:

- CIM_ClassCreation
- CIM_ClassDeletion
- CIM_ClassModification
- CIM_InstCreation
- CIM_InstModification
- CIM_InstDeletion
- CIM_InstRead

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

**Note** – In most cases, providers should handle their own indications because polling carries a high overhead. In order to generate indications, the provider must poll. In this case, the provider can delegate the task to the CIMOM.

If a provider implements the EventProvider interface, the CIMOM invokes the methods in the interface and takes actions according to the responses. When the CIMOM 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 whether 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 CIMOM to determine whether the subscription is authorized. The provider can make the determination based on either the user ID of the owner of the indication handler, which is the user who receives the indications, or 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 name space.

   If the provider does not implement the EventProvider interface and the CIMOM tries to poll, the authorization succeeds if enumerateInstances succeeds on the provider.
3. **activateFilter**– Invoked by the CIMOM 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 CIMOM. For example, if the destination handler malfunctions.

### About Subscriptions

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

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

- The indications to which the application wants to subscribe
- The handler to which the CIMOM delivers the indication

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

### To Create a Subscription

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

1. Create an instance of **CIM Listener**. See [Adding a CIM Listener](#).

2. Create an instance of **CIM_IndicatorFilter**. See [Creating an Event Filter](#).

3. Create an instance of **CIM_IndicatorHandler**. See [Creating an EventHandler](#).

4. Bind the **CIM_IndicatorFilter** to the **CIM_IndicatorHandler**. See [Binding an Event Filter to an Event Handler](#).

### Adding a CIM Listener

To receive indications of CIM events, first add an instance of **CIMListener** to **CIMClient**, by invoking the `addCIMListener` method on **CIMClient**.
Note – The CIMListener interface must implement the indicationOccurred method, which takes the argument CIMEvent. This method is invoked when an indication is available for delivery.

EXAMPLE 4-21 Adding a CIM Listener

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

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

Creating an Event Filter

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

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemCreationClassName</td>
<td>The name of the system on which the creation class for the filter resides or to which it applies.</td>
<td>Optional. The value is decided by the CIM Object Manager.</td>
</tr>
<tr>
<td>SystemName</td>
<td>The name of the system on which the filter resides or to which it applies.</td>
<td>Optional. The default for this key property is the name of the system on which the CIM Object Manager is running.</td>
</tr>
<tr>
<td>CreationClassName</td>
<td>The name of the class or subclass used to create the filter.</td>
<td>Optional. The CIM Object Manager assigns CIM_IndicationFilter as the default for this key property.</td>
</tr>
<tr>
<td>Name</td>
<td>The unique name of the filter.</td>
<td>Optional. The CIM Object Manager assigns a unique name.</td>
</tr>
</tbody>
</table>
TABLE 4–6  CIM_IndicationFilter Properties  (Continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceNamespace</td>
<td>The path to a local name space where the CIM indications originate.</td>
<td>Optional. The default is null.</td>
</tr>
<tr>
<td>Query</td>
<td>A query expression that defines the conditions under which indications are generated. Currently, only Level 1 WBEM Query Language (WQL) expressions are supported. To learn more about WQL query expressions, see Chapter 5, “Writing WBEM Queries.”</td>
<td>Required.</td>
</tr>
<tr>
<td>QueryLanguage</td>
<td>The language in which the query is expressed.</td>
<td>Required. The default is WQL (WBEM Query Language).</td>
</tr>
</tbody>
</table>

▼ To Create an Event Filter

1. Create an instance of the CIM_IndicationFilter class

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

2. Specify the name of the event filter

   ```java
   Name = "filter_all_new_solarisdiskdrives"
   ```

3. Create a WQL string identifying event indications to return

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

4. Set property values in the cimfilter instance to identify the following information:

   - Name of the filter
   - Filter string to select CIM events
   - Query language (WQL) to parse the query string

   ```java
   ci.setProperty("Name", new CIMValue("filter_all_new_solarisdiskdrives"));
   ci.setProperty("Query", new CIMValue(filterString));
   ci.setProperty("QueryLanguage", new CIMValue("WQL"));
   ```
5 Create a `cimfilter` instance that is called `filter`. Store the instance in the CIM Object Manager Repository

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

### Example 4–22 Creating an Event Filter

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

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

### Creating an Event Handler

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required or Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemCreationClassName</td>
<td>The name of the system on which the creation class for the handler resides or to which it applies.</td>
<td>Optional. Completed by the CIM Object Manager.</td>
</tr>
<tr>
<td>SystemName</td>
<td>The name of the system on which the handler resides or to which it applies.</td>
<td>Optional. The default value for this key property is the name of the system on which the CIM Object Manager is running.</td>
</tr>
<tr>
<td>CreationClassName</td>
<td>The class or subclass used to create the handler.</td>
<td>Optional. The CIM Object Manager assigns the appropriate class name as the default for this key property.</td>
</tr>
<tr>
<td>Name</td>
<td>The unique name of the handler.</td>
<td>Optional. The client application must assign a unique name.</td>
</tr>
</tbody>
</table>
### TABLE 4-7  CIM IndicationHandler Properties (Continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required or Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>The name of the entity that created or maintains this handler. The provider can check this value to determine whether to authorize a handler to receive an indication.</td>
<td>Optional. The default value is the Solaris OS user name of the user who creates the instance.</td>
</tr>
</tbody>
</table>

EXAMPLE 4-23  Creating an EventHandler

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

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

### Binding an Event Filter to an Event Handler

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

The following example creates a subscription (filterdelivery) and defines the filter property to the filter object path created in "To Create an Event Filter" on page 76. The example also defines the handler property to the delivery object path that is created in Example 4-23.

EXAMPLE 4-24  Binding an Event Filter to an Event Handler

```java
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);
```
Reading and Writing Log Messages

The Solaris platform MOF files include logging classes. Clients can create and read log records using these classes to record errors, warnings, and informational messages. For example, a log message can indicate one of the following conditions:

- A system cannot access a serial port
- A system successfully mounts a file system
- The number of processes that are running on a system exceeds the allowed number

The underlying providers for the logging classes can forward logging requests to the syslog daemon, the default logging system in the Solaris OS. See the \texttt{syslogd(1M)} man page for more information.

About Log Files

WBEM log messages are stored in individual log files in the \texttt{/var/sadm/wbem/log} directory. Properties that you manipulate with the singleton instance of the \texttt{Solaris\_LogServiceProperties} class:

- Names of the log files
- Directory in which the log files are stored
- Log file size limit
- Number of log files to store
- Whether to forward messages to \texttt{syslogd(1M)}

The format of each log entry is defined by the \texttt{Solaris\_LogEntry} class, which is a subclass of \texttt{CIM\_LogRecord}. You can find \texttt{Solaris\_LogEntry} in \texttt{Solaris\_Device.mof}, and \texttt{CIM\_LogRecord} in \texttt{CIM\_Device26.mof}.

A log message includes the following elements:

\textbf{TABLE 4-8 Log Message Elements}

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Type of message – application, system, or security</td>
</tr>
<tr>
<td>Severity</td>
<td>Severity of the condition – warning or error</td>
</tr>
<tr>
<td>Application</td>
<td>Name of the application or the provider that is writing the log message</td>
</tr>
<tr>
<td>User</td>
<td>Name of the user who was using the application when the log message was generated</td>
</tr>
<tr>
<td>Client Machine</td>
<td>Name and IP address of the system that the user was on when the log message was generated</td>
</tr>
</tbody>
</table>
The following examples show how to create a log and how to display the contents of a log.

**EXAMPLE 4-25 Creating an Instance of Solaris_LogEntry**

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

```java
public class CreateLog {
    public static void main(String args[]) throws CIMException {
        // Display usage statement if insufficient command line
        // arguments are passed.
        if (args.length < 3) {
            System.out.println("Usage: CreateLog host username password
            + [rmi|http]");
            System.exit(1);
        }
        String protocol = CIMClient.CIM_RMI;
        CIMClient cc = null;
        CIMObjectPath cop = null;
        BufferedReader d = new BufferedReader(new InputStreamReader(System.in));
        String input_line = "";
        // Query user for number of records that need to be created.
        System.out.print("How many log records do you want to write? ");
        int num_recs = 0;
        try {
            num_recs = Integer.parseInt(d.readLine());
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(1);
        }
        // Over-arching try-catch block
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal(args[1]);
```
PasswordCredential pc = new PasswordCredential(args[2]);

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

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

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

// Prompt user for relevant info needed to create the
// log record.
System.out.println("Please enter the record Category: ");
System.out.println("\t(0)application, (1)security, 
(2)system");
logsvcKey = new CIMProperty("category");
input_line = d.readLine();
logsvcKey.setValue(new CIMValue(Integer.valueOf(input_line)));
keys.addElement(logsvcKey);
System.out.println("Please enter the record Severity:");
System.out.println("\t(0)Informational, (1)Warning, 
(2)Error");
logsvcKey = new CIMProperty("severity");
input_line = d.readLine();
logsvcKey.setValue(new CIMValue(Integer.valueOf(input_line)));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("Source");
System.out.println("Please enter Application Name:");
logsvcKey.setValue(new CIMValue(d.readLine()));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SummaryMessage");
System.out.println("Please enter a summary message:");
logsvcKey.setValue(new CIMValue(d.readLine()));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("DetailedMessage");
System.out.println("Please enter a detailed message:");
logsvcKey.setValue(new CIMValue(d.readLine()));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("RecordData");
logsvcKey.setValue(new CIMValue("0xfe 0x45 0xae 0xda random data"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SyslogFlag");
logsvcKey.setValue(new CIMValue(new Boolean(true)));
keys.addElement(logsvcKey);
CIMObjectPath logreccop = new CIMObjectPath("Solaris_LogEntry", keys);
CIMClass logClass = cc.getClass(logreccop);
CIMInstance ci = logClass.newInstance();
Ci.setClassName("Solaris_LogEntry");
Creating an Instance of Solaris_LogEntry  

(Continued)

```java
ci.getProperties(keys);
// System.out.println(ci.toString());

// Create as many instances of the record as requested.
for (int i = 0; i < num_recs; i++) {
    cc.createInstance(logreccop, ci);
}
``` 

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

} 

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

EXAMPLE 4–26  Displaying a List of Log Records

This example displays a list of log records.

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

        String protocol = CIMClient.CIM_RMI;

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

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

        // Over-arching try-catch block
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal(args[1]);
            PasswordCredential pc = new PasswordCredential(args[2]);

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

            cc = new CIMClient(cns, up, pc, protocol);
```
EXAMPLE 4–26  Displaying a List of Log Records  (Continued)

cop = new CIMObjectPath("Solaris_LogEntry");

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

// iterate over the list and print out each property.
for (; e.hasMoreElements(); ) {
    System.out.println("----------------------------------");
    CIMInstance ci = (CIMInstance)e.nextElement();
    System.out.println("Log filename : "+
        ((String)ci.getProperty("LogName").getValue()).
        getValue());
    int categ =
        ((Integer)ci.getProperty("Category").getValue()).
        getValue());
    if (categ == 0)
        System.out.println("Category : Application Log");
    else if (categ == 1)
        System.out.println("Category : Security Log");
    else if (categ == 2)
        System.out.println("Category : System Log");
    int severity =
        ((Integer)ci.getProperty("Severity").getValue()).
        getValue());
    if (severity == 0)
        System.out.println("Severity : Informational");
    else if (severity == 1)
        System.out.println("Severity : Warning Log!");
    else if (severity == 2)
        System.out.println("Severity : Error!!");
    System.out.println("Log Record written by : "+
        ((String)ci.getProperty("Source").getValue()).
        getValue()));
    System.out.println("User : "+
        ((String)ci.getProperty("UserName").getValue()).
        getValue()));
    System.out.println("Client Machine : "+
        ((String)ci.getProperty("ClientMachineName").getValue()).
        getValue()));
    System.out.println("Server Machine : "+
        ((String)ci.getProperty("ServerMachineName").getValue()).
        getValue()));
    System.out.println("Summary Message : "+
        ((String)ci.getProperty("SummaryMessage").getValue()).
        getValue()));
    System.out.println("Detailed Message : "+
        ((String)ci.getProperty("DetailedMessage").getValue()).
        getValue()));
    System.out.println("Additional data : "+
        ((String)ci.getProperty("RecordData").getValue()).
        getValue()));
    boolean syslogflag =
        ((Boolean)ci.getProperty("SyslogFlag").getValue()).
        getValue());
    if (syslogflag == true) {
        System.out.println("Record was written to syslog");
    } else {
        System.out.println("Record was not written to syslog");
    }
    System.out.println("----------------------------------");
}  
} catch (Exception e) {  
}
EXAMPLE 4–26  Displaying a List of Log Records  (Continued)

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

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

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

- “About the WBEM Query Language” on page 85
- “Writing Queries” on page 86
- “Parsing Queries” on page 89

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

### About the WBEM Query Language

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

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

<table>
<thead>
<tr>
<th>SQL Concept</th>
<th>WQL Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>CIM class</td>
</tr>
<tr>
<td>Row</td>
<td>CIM instance</td>
</tr>
<tr>
<td>Column</td>
<td>CIM property</td>
</tr>
</tbody>
</table>
Note – Like SQL, WQL statements use single (’’) quotation marks.

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

Writing Queries

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

The following example shows a search for all instances of the Solaris_DiskDrive class that have a particular value for the Storage_Capacity property:

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

WQL Key Words

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

<table>
<thead>
<tr>
<th>Key Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Combines two Boolean expressions and returns TRUE when both expressions are TRUE.</td>
</tr>
<tr>
<td>FROM</td>
<td>Specifies the classes that contain the properties that are listed in a SELECT statement.</td>
</tr>
<tr>
<td>NOT</td>
<td>Comparison operator that is used with NULL.</td>
</tr>
<tr>
<td>OR</td>
<td>Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Specifies the properties that are used in a query.</td>
</tr>
<tr>
<td>WHERE</td>
<td>Narrows the scope of a query.</td>
</tr>
<tr>
<td>LIKE</td>
<td>Generates a result set that is based on a minimum amount of information that was provided.</td>
</tr>
</tbody>
</table>
**SELECT Statement**

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

*Note* – Currently, join operations are not supported.

The syntax for the SELECT statement is as follows:

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

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

### TABLE 5–3  Sample SELECT Statements

<table>
<thead>
<tr>
<th>Example Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM class</td>
<td>Selects all instances of the specified class and all of its subclasses. Each instance that is returned contains all the properties.</td>
</tr>
<tr>
<td>SELECT PropertyA FROM class</td>
<td>Selects all instances that contain PropertyA of the specified class and all of its subclasses.</td>
</tr>
<tr>
<td>SELECT PropertyA, PropertyB FROM class WHERE PropertyB=20</td>
<td>Selects all instances of the specified class and all of its subclasses where PropertyB=20. Each returned instance contains only PropertyA and PropertyB.</td>
</tr>
</tbody>
</table>

**FROM Clause**

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

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

**WHERE Clause**

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

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

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

The `conditional_expression` in the WHERE clause takes the following form:
property operator constant

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

```
SELECT instance FROM class WHERE constant operator property
```

Valid WHERE clauses follow these rules:

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

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

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

  The following 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'
  ```

  The following example retrieves disks named home and files only if the disks have a certain amount of available space remaining, and have Solaris platform file systems.

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

**Standard WQL Operators for WHERE Clauses**

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

**TABLE 5–4** WQL Operators for WHERE Clauses

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>
Parsing Queries

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

<table>
<thead>
<tr>
<th>Query Part</th>
<th>Accessor Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT list</td>
<td>getSelectList</td>
</tr>
<tr>
<td>FROM clause</td>
<td>getFromClause</td>
</tr>
<tr>
<td>WHERE clause</td>
<td>getWhereClause</td>
</tr>
</tbody>
</table>

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

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

### SELECT List

The list returned by the `getSelectList` method for each `SelectExp` is an instance of the `SelectList` class. This list specifies properties, as `AttributeExp` instances, that must be included in the selected instances. You can retrieve these `AttributeExp` instances by using the `elements` method of `SelectList`. Each attribute denotes the name of a column that maps to a property of a `CIMInstance` in WQL. The `AttributeExp` has an `apply` method that, when passed in a `CIMInstance`, returns the value of the property that the `AttributeExp` represents. The `SelectList` has an `apply` method that, when passed in a `CIMInstance`, returns a `CIMInstance` containing only the properties that the `SelectList AttributeExp` instances denote.

### FROM Clause

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

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

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

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

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

The QueryExp has two other useful methods, canonize00C and canonize00D, which are used to simplify conditional expressions for further processing. The canonize00C method converts the parse tree from an arbitrary combination of ANDs and ORs to a canonical disjunction of conjunctions form (OR of ANDs). The canonize00D method converts the parse tree from an arbitrary combination of ANDs and ORs to canonical conjunction of disjunctions form (AND of ORs). These classes and methods are used by providers that need to filter instances based on input queries.

Note – Details of these classes can be found in the API reference pages, generated by the javadoc command. See file:/usr/sadm/lib/wbem/doc/index.html.

Writing a Provider That Handles Queries

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

EXAMPLE 5-1  Provider That Handles Queries

/*
 * The execQuery method will support only limited queries
 * based upon partial key matching. An empty Vector is
public CIMInstance[] execQuery(CIMObjectPath op, String query, String ql, CIMClass cc) throws CIMException {
    Vector result = new Vector();
    try {
        SelectExp q = new SelectExp(query);
        SelectList attrs = q.getSelectList();
        NonJoinExp from = (NonJoinExp)q.getFromClause();
        QueryExp where = q.getWhereClause();

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

        // filtering the instances
        for (int i = 0; i < v.length; i++) {
            if ((where == null) || (where.apply(v[i]) == true)) {
                result.addElement(attrs.apply(v[i]));
            }
        }
    } catch (Exception e) {
        throw new CIMException(CIMException.CIM_ERR_FAILED, e.toString());
    }
    return (CIMInstance[])result.toArray();
} // execQuery

EXAMPLE 5–1 Provider That Handles Queries (Continued)

* returned if no entries are selected by the query.
* @param op The CIM object path of the CIM instance to be returned
* @param query The CIM query expression
* @param ql The CIM query language indicator
* @param cc The CIM class reference
* @return A vector of CIM object instances
* @version 1.19 01/26/00
* @author Sun Microsystems, Inc.
*/
Writing a Provider Program

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

- “About Providers” on page 93
- “Implementing the Provider Interfaces” on page 96
- “Creating a Provider” on page 106

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

About Providers

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

Solaris software providers exist for a variety of areas:

- Users
- Groups
- Aliases
- Roles
- File systems
- Disks
- Processes
- The cron tool
About Providers

- Network configuration
- Product registry
- Device and system performance monitoring

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

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

Providers perform the following tasks:

- **Provide data to management applications** – When a management application requests data about a managed resource that is not available in the CIM Object Manager Repository, the CIMOM forwards the request to a provider. The provider accesses the data from the managed resource and passes the data back to the CIMOM. If the data received from a managed resource is in a native format such as C code, the provider maps the data to Java CIM classes prior to passing the data to the CIMOM.

- **Control management resources** – When a management application sends data to the CIMOM to control a managed resource, the CIMOM passes the data to the appropriate provider. If the managed resource requires data in a native format, the provider maps the CIM classes to the resource’s native format prior to passing the data along.

*Note* – Providers must reside on the same machine as the CIMOM.

**Provider Data Sources**

Providers can retrieve data from the following sources:

- **Non-persistent data** – Variables that are local to the provider class that exist only when the provider’s methods are run.

- **Persistent memory that is local to the provider** – Used by creating global variables in the provider class. This provider memory is erased when the CIMOM is stopped and restarted.

- **CIM Object Manager Repository** – This persistent memory is erased when Solaris WBEM Services software is uninstalled. The provider must use CIMOM handles and an internal provider to access this memory through the CIMOM.

- **Files and databases maintained by the provider, or dynamic data** – Providers can generate data dynamically by retrieving data from a system. For example, a provider can make a system call to retrieve the number of processes currently running.
Providers are categorized according to the types of requests the providers handle. Client programs communicate with the CIMOM and access WBEM data through the client API. The CIMOM maps the provider methods to the corresponding client methods in the client API. However, the argument lists and return values of corresponding methods might differ. See file:/usr/sadm/lib/wbem/doc/index.html.

- If a provider stores data in the CIM Object Manager Repository, then the provider accesses the Repository using handles to the CIMOM. These handles call the methods of the client API. See “Implementing the Provider Interfaces” on page 96.
- If a provider needs to create instances or associations in the CIM Object Manager Repository, then it uses an internal provider. The provider calls methods of Instance or Associator Providers that are internal to WBEM. See "Implementing the Provider Interfaces" on page 96.

Ensure that your argument list and return type are correct for the method and class used.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>ClassName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>CIMInstanceProvider</td>
<td>Supply dynamic instances of a given class, and support instance retrieval,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enumeration, modification, and deletion</td>
</tr>
<tr>
<td>Method</td>
<td>MethodProvider</td>
<td>Supply methods of one or more classes</td>
</tr>
<tr>
<td>Associator</td>
<td>CIMAssociatorProvider</td>
<td>Supply instances of dynamic association classes</td>
</tr>
<tr>
<td>Indication</td>
<td>EventProvider</td>
<td>Handle indications of CIM events</td>
</tr>
<tr>
<td>Authorizable</td>
<td>None</td>
<td>A marker interface that indicates to the CIMOM that the provider does its</td>
</tr>
<tr>
<td></td>
<td></td>
<td>own authorization check</td>
</tr>
</tbody>
</table>

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

Provider Naming Conventions

You can include providers in a single Java class file or store each provider in a separate class. The provider name identifies the Java class that serves as the provider for the class. Currently, the CIMOM supports only providers that are written in the Java language.
Provider and class names must follow these rules:

- The class name must be a valid CIM class. The name must contain a prefix of characters, followed by an underscore, followed by more characters.
  For example, `green_apples` and `red_apples` are valid CIM class names, whereas `apples`, `apples_`, and `_apples` are not.
- The provider name that is specified in the MOF file must match the name of the provider class file.
  For example, `SimpleCIMInstanceProvider` is the provider and `Ex_SimpleCIMInstanceProvider` is the class.

**Note** – You must prepend “java:” to every provider qualifier to notify the CIMOM that the provider is written in the Java language.

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

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

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

**Implementing the Provider Interfaces**

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

- `initialize(CIMOMHandle cimom)` – If your provider stores data in the CIM Object Manager Repository, it must assign the passed CIMOM handle to the CIMOM handle that it will use to contact the CIMOM. These requirements are illustrated in the following example.

```java
private CIMOMHandle cimom = null;
...
public void initialize(CIMOMHandle cimom)
      throws CIMException {
      this.cimom = (CIMOMHandle) cimom;
```
Before the provider can create instances or manipulate associations in the CIM Object Manager Repository, the provider must first cast the passed CIMOM handle to the subclass ProviderCIMOMHandle. Then the provider must fetch an internal instance or association provider. The following example illustrates how to perform these tasks.

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

**Note** – The `initialize` command automatically runs each time a provider is initialized after the CIMOM restarts.

- `cleanup` – Currently acts as a placeholder.

## Writing an Instance Provider

This following sample code implements the `enumerateInstances` and `getInstance` methods of the `Ex_SimpleCIMInstanceProvider` class. For brevity, the implementations of the `deleteInstance`, `createInstance`, `setInstance`, and `execQuery` methods in this example just throw a `CIMException`.

**Note** – For information on implementing the `execQuery` method, see “Parsing Queries” on page 89.

```
EXAMPLE6-1  CIMInstance Provider
/*
 * SimpleCIMInstanceProvider.java
 */
import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.provider.CIMProvider;
import javax.wbem.provider.CIMInstanceProvider;
import javax.wbem.provider.MethodProvider;
import java.util.*;
import java.io.*;

public class SimpleCIMInstanceProvider implements CIMInstanceProvider {
    static int loop = 0;
    public void initialize(CIMOMHandle cimom) throws CIMException {
        // Implementation...
    }
    public void cleanup() throws CIMException {
        // Implementation...
    }
    public CIMObjectPath[] enumerateInstanceNames(CIMObjectPath op,
```
EXAMPLE 6-1  CIMInstance Provider  (Continued)

```java
public CIMInstance[] enumerateInstances(CIMObjectPath op,
  boolean localOnly, boolean includeQualifiers,
  boolean includeClassOrigin, String[] propertyList,
  CIMClass cc) throws CIMException {
  if (op.getObjectName().equalsIgnoreCase("Ex_SimpleCIMInstanceProvider")) {
    Vector instances = new Vector();
    CIMInstance ci = cc.newInstance();
    if (loop == 0) {
      ci.setProperty("First", new CIMValue("red"));
      ci.setProperty("Last", new CIMValue("apple"));
      // only include the properties that were requested
      ci = ci.filterProperties(propertyList, includeQualifier,
                                includeClassOrigin);
      instances.addElement(ci);
      loop += 1;
    } else {
      ci.setProperty("First", new CIMValue("red"));
      ci.setProperty("Last", new CIMValue("apple"));
      // only include the requested properties
      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 requested properties
      ci = ci.filterProperties(propertyList, includeQualifier,
                                includeClassOrigin);
      instances.addElement(ci);
    }
    return (CIMInstance[]) instances.toArray();
  } else {
    throw new CIMException(CIM_ERR_INVALID_CLASS);
  }
}
```

Implementing the Provider Interfaces

---

Solaris WBEM Developer's Guide  •  January 2005
EXAMPLE 6-1 CIMInstance Provider  (Continued)

// 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;

}  

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

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

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

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


Writing a Method Provider

The method invokeMethod is the only way that a client program can call the methods of Solaris WBEM providers. This condition is true for both providers that are built in or that are added by developers.

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

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

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

**EXAMPLE 6-2** Method Provider

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

    public void cleanup() throws CIMException {
        // cleanup code
    }

    public CIMValue invokeMethod(CIMObjectPath op, String methodName,
                                  Vector inParams, Vector outParams) throws CIMException {
        if (op.getObjectName().equalsIgnoreCase("solaris_computersystem")) {
            if (methodName.equalsIgnoreCase("reboot")) {
                // call helper function, not shown here
                return new CIMValue(rebootSystem());
            }
            if (methodName.equalsIgnoreCase("shutdown")) {
                // call helper function, not shown here
                return new CIMValue(shutdownSystem());
            }
        }
        if (op.getObjectName().equalsIgnoreCase("solaris_serialport")) {
            if (methodName.equalsIgnoreCase("disableportservice")) {
                // call helper function, not shown here
                return new CIMValue(deletePort(op));
            }
        }
        throw new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED,
                                "The requested function does not exist");
    }
    // helper functions would be defined below

    // helper functions would be defined below
}
```
Writing an Associator Provider

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

Unless the CIMOM can locate the object path of an instance, it assumes that the client wants the class definitions of the association that are in the CIM Object Manager Repository. The class definition of the association includes the templates from which the association’s member instances are derived. Therefore, the CIMOM uses the client API’s association method and not the provider’s association method.

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

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

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

- A one-to-many relationship:

```java
class Classroom
{
    Teacher REF teaches;
    Student1 REF taughtby;
    Student2 REF taughtby;
    Student3 REF taughtby;
    Student4 REF taughtby;
};
```

- A many-to-many relationship:

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

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

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

```
EXAMPLE 6–3  CIMASSOCIATOR Provider

   public CIMInstance[] associators(CCIMObjectPath assocName, CIMObjectPath objectName, String resultClass, String role, String resultRole, boolean includeQualifiers, boolean includeClassOrigin, String[] propertyList)
   throws CIMException {
   System.out.println("Associators "+assocName+" "+objectName);
   if (objectName.getObjectName().equalsIgnoreCase("example_teacher")) {
      Vector v = new Vector();
      if ((role != null) && (!role.equalsIgnoreCase("teaches"))) {
         // Teachers only play the teaches role.
         return v;
      }
      if ((resultRole != null) && (!resultRole.equalsIgnoreCase("taughtby"))) {
         // Teachers only result in taughtby role
         return v;
      }
      // Get the associators of a teacher
      CIMProperty nameProp = (CIMProperty)objectName.getKeys().elementAt (0);
      String name = (String)nameProp.getValue().getValue();
      // Get the student class
      CIMObjectPath tempOp = new CIMObjectPath("example_student");
      tempOp.setNameSpace(assocName.getNameSpace());
      CIMClass cc = cimom.getClass(tempOp, false);
      // Test the instance name passed by objectName
      // and return the associated instances of the student class.
      if(name.equalsIgnoreCase("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;
      }
```

Implementing the Provider Interfaces
EXAMPLE 6–3  CIM Associator Provider  (Continued)

Writing an Indication Provider

To generate an indication for a CIM event, you need to perform the following tasks:

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

▼ How To Generate an Event Indication

1  **Implement the EventProvider interface.**
   
   For example:

   ```java
   public class sampleEventProvider implements InstanceProvider EventProvider{ }
   ```

2  **Execute each of the methods listed in Table 6–2 for each instance indication that the provider handles.**

3  **Create an indication for create, modify, and delete instance event type.**
   
   For example, in the createInstance method:

   ```java
   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);
   }
   ```
Deliver the event indication to the CIM Object Manager.

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

**Event Provider Methods**

An event provider implements the `EventProvider` interface. This interface contains methods that the CIMOM uses to notify the provider when a client has subscribed for indications of CIM events. This method is also used when a client has cancelled the subscription for CIM events. These methods allow the provider to indicate whether the CIMOM should poll for some event indications and whether the provider should authorize the return of an indication to a handler.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateFilter</td>
<td>When a client creates a subscription, the CIMOM calls this method to ask the provider to check for CIM events.</td>
</tr>
<tr>
<td>authorizeFilter</td>
<td>When a client creates a subscription, the CIMOM calls this method to test if the specified filter expression is allowed.</td>
</tr>
<tr>
<td>deActivateFilter</td>
<td>When a client removes a subscription, the CIMOM calls this method to ask the provider to deactivate the specified event filter.</td>
</tr>
<tr>
<td>mustPoll</td>
<td>When a client creates a subscription, the CIMOM calls this method to test whether the specified filter expression is allowed by the provider, and if it must be polled.</td>
</tr>
</tbody>
</table>

The CIMOM 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 filter is the first filter for this event type. The `deActivateFilter` method takes the boolean `lastActivation`, indicating that this filter is the last filter for this event type.
Creating and Delivering Indications

A client application subscribes for indications of CIM events by creating an instance of the CIM_IndicationSubscription class. The CIMOM then forwards the request to the appropriate provider. If the provider implements the EventProvider interface, the CIMOM notifies the provider when to start sending indications for the specified events. The provider performs this notification by calling the provider's activateFilter method. In addition, the CIMOM notifies the provider when to stop sending indications for the specified events by calling the provider's deActivateFilter method.

The provider responds to the CIMOM'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 CIMOM calls the activateFilter method. This flag is cleared when the CIMOM 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. The provider uses the deliverEvent method to return the indication to the CIMOM. 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 ProviderCIMOMHandle.deliverIndication method. The CIMOM receives the indication and delivers the indication to the appropriate indication handlers. A provider can handle multiple event types. For example, in the case of life cycle indications, a provider can handle CIM_InstCreation, CIM_InstDeletion, and CIM_InstModification.

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

About Authorizations

A provider that handles sensitive data can check authorizations for requests for indications. The provider must implement the Authorizable interface to indicate that the provider handles authorization checking. The provider also implements the authorizeFilter method. The CIMOM calls this method to test whether 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, the event handler, can be different than the owner of the client application requesting the filter activation.
Writing a Native Provider

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

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

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

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

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

Creating a Provider

Follow these steps to create a provider:

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

▼ How to Set the Provider CLASSPATH

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

1. Create an instance of the Solaris_ProviderPath class.
   For example:
   ```java
   /* Create a name space object initialized with root\system (name of name space) on the local host. */
   CIMNameSpace cns = new CIMNameSpace("", "root\system");
   ```
// Connect to the root\system name space as root.
c = new CIMClient(cns, "root", "root_password");

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

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

2 Set the pathurl property to the location of the files by using standard URL format.
   For example:
   /* Set the provider CLASSPATH to /myhome/myprovider */
   ci.setProperty("pathurl", new CIMValue(new String
   ("file:///myhome/myprovides")));

   The standard URL format is shown in the following table.

<table>
<thead>
<tr>
<th>Provider CLASSPATH</th>
<th>Standard URL Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute path to directory</td>
<td>file:///a/b/c/</td>
</tr>
<tr>
<td>Absolute path to .jar file</td>
<td>file:///a/b/my.jar</td>
</tr>
</tbody>
</table>

3 Create the instance.
   For example:
   // Pass the updated instance to the CIM Object Manager
   cc.createInstance(new CIMObjectPath(), ci);

▼ How to Register a Provider

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

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

Note – For more information on creating MOF files, see the DMTF Web site at http://www.dmtf.org.
2 Include the provider qualifier in the MOF file to specify the provider type and location for the CIMOM.

For example:

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

This qualifier indicates the following information:

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

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

Example 6–4 Registering a Provider

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

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

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

// ........................................................................
// SimpleCIMInstanceProvider
// ........................................................................
[Provider("java:SimpleCIMInstanceProvider")]
class Ex_SimpleCIMInstanceProvider {
  // Properties
  [Key, Description("First Name of the User")]
  string First;
  [Description("Last Name of the User")]
  string Last;
};
```
Creating JavaBeans Components Using the MOF Compiler

This chapter provides an overview of the Managed Object Format (MOF) compiler. This chapter also describes how to create JavaBeans components by using the -j option to the mofcomp command. This chapter covers the following topics.

- "About the MOF Compiler" on page 109
- "How CIM Maps to the Java Programming Language" on page 111
- "Example of Generating JavaBeans Components" on page 115

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

About the MOF Compiler

Managed Object Format (MOF) is a compiled language developed by the Distributed Management Task Force (DMTF). The MOF language defines static and dynamic classes and instances for CIM and WBEM. You can use the CIM and Solaris platform MOF files that are included with Solaris WBEM Services. You can also create your own MOF files. For more information on creating your own MOF files using the DMTF’s MOF language, see the DMTF Web site at http://www.dmtf.org.

The MOF Compiler, mofcomp(1M) performs the following tasks:

- Parses MOF files
- Converts the classes and instances to Java programming language classes
- Adds the classes to the CIM Object Manager Repository in the default (root\cimv2) or other specified name space

You can easily convert MOF files to the Java programming language. As a result, applications based on Java technology can interpret and exchange data in MOF files on any machine that runs a Java Virtual Machine.
During a Solaris OS installation, the MOF compiler compiles the bundled MOF files that describe the CIM and Solaris platform schema and adds those files to the CIM Object Manager Repository.

**Generating JavaBeans Components Using mofcomp**

In the context of WBEM, JavaBeans components, or beans, define methods for accessing and manipulating CIM classes and data. To simplify your development efforts, you can use the -j option to the mofcomp command to generate beans from the CIM classes in your MOF files. These automatically-generated beans define the interfaces. You must add the implementation code.

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

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

For each CIM class that is defined in a MOF file, the MOF compiler JavaBeans generation feature generates a Java programming language interface that contains the following methods:

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

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

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

**MOF File Elements**

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

TABLE 7–1 MOF File Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKAGE</td>
<td>Required. Specifies the name of the Java package that contains the source files generated by the MOF compiler.</td>
</tr>
<tr>
<td>IMPORTS</td>
<td>Optional. Specifies the names of the Java classes to import into the generated source files. These classes are separated with a colon(:). You can specify as many Java classes as you want, on as many lines as you want.</td>
</tr>
<tr>
<td>EXCEPTIONS</td>
<td>Optional. Specifies the names of the Java exceptions that are included in the generated source files. These exceptions are separated with a colon(:). You can specify as many Java class exceptions as you want, on as many lines as you want.</td>
</tr>
</tbody>
</table>

Note – If you specify EXCEPTIONS, you must specify IMPORTS.

How CIM Maps to the Java Programming Language

The following table describes how CIM elements map to elements of the Java programming language.

TABLE 7–2 How CIM Elements Map to Java Elements

<table>
<thead>
<tr>
<th>CIM Element</th>
<th>Java Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>The CIM class name is used as the basis for the name of the generated Java source files. The generated Java classes follow the same inheritance as defined in the class-subclass relationships in the MOF.</td>
</tr>
<tr>
<td>Property</td>
<td>An accessor and a mutator method are created for each CIM property. The CIM property name is used as the basis for the associated accessor and mutator methods.</td>
</tr>
<tr>
<td>Method</td>
<td>For each CIM method, a comparable Java method is created. The method name is used as the basis for the related Java method name. The return value is the same, accounting for the mapping to a Java data type. Input and output parameters are used as arguments to the Java method. Output parameters are not directly included in the method signature. Instead, output parameters are encapsulated in an output container object that is included as a method parameter.</td>
</tr>
<tr>
<td>Qualifier</td>
<td>Qualifiers are described in Table 7–4 and Table 7–5.</td>
</tr>
<tr>
<td>Association</td>
<td>Nothing specific required.</td>
</tr>
</tbody>
</table>
TABLE 7–2  How CIM Elements Map to Java Elements  (Continued)

<table>
<thead>
<tr>
<th>CIM Element</th>
<th>Java Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indication</td>
<td>Nothing specific required.</td>
</tr>
<tr>
<td>Reference</td>
<td>For each CIM reference, a reference to a generated Java interface is created.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Nothing specific required.</td>
</tr>
<tr>
<td>Schema</td>
<td>Nothing specific required.</td>
</tr>
</tbody>
</table>

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

TABLE 7–3  How CIM Data Types Map to Java Data Elements

<table>
<thead>
<tr>
<th>CIM Data Type</th>
<th>Java Data Type</th>
<th>Accessor Method</th>
<th>Mutator Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 X</td>
<td>UnsignedInt8</td>
<td>UnsignedInt8 getX();</td>
<td>void setX(UnsignedInt8 x);</td>
</tr>
<tr>
<td>sint8 X</td>
<td>Byte</td>
<td>Byte getX();</td>
<td>void setX(Byte x);</td>
</tr>
<tr>
<td>uint16 X</td>
<td>UnsignedInt16</td>
<td>UnsignedInt16 getX();</td>
<td>void setX(UnsignedInt16 x);</td>
</tr>
<tr>
<td>sint16 X</td>
<td>Short</td>
<td>Short getX();</td>
<td>void setX(Short x);</td>
</tr>
<tr>
<td>uint32 X</td>
<td>UnsignedInt32</td>
<td>UnsignedInt32 getX();</td>
<td>void setX(UnsignedInt32 x);</td>
</tr>
<tr>
<td>sint32 X</td>
<td>Integer</td>
<td>Integer getX();</td>
<td>void setX(Integer x);</td>
</tr>
<tr>
<td>uint64 X</td>
<td>UnsignedInt64</td>
<td>UnsignedInt64 getX();</td>
<td>void setX(UnsignedInt64 x);</td>
</tr>
<tr>
<td>sint64 X</td>
<td>Long</td>
<td>Long getX();</td>
<td>void setX(Long x);</td>
</tr>
<tr>
<td>String X</td>
<td>String</td>
<td>String getX();</td>
<td>void setX(String x);</td>
</tr>
<tr>
<td>Boolean X</td>
<td>Boolean</td>
<td>Boolean isX();</td>
<td>void setX(Boolean x);</td>
</tr>
<tr>
<td>real32 X</td>
<td>Float</td>
<td>Float getX();</td>
<td>void setX(Float x);</td>
</tr>
<tr>
<td>real64 X</td>
<td>Double</td>
<td>Double getX();</td>
<td>void setX(Double x);</td>
</tr>
<tr>
<td>DateTime X</td>
<td>CIMDateTime</td>
<td>CIMDateTime getX();</td>
<td>void setX(CIMDateTime x);</td>
</tr>
<tr>
<td>Reference X</td>
<td>CIMObjectPath</td>
<td>CIMObjectPath getX();</td>
<td>void setX(CIMObjectPath x);</td>
</tr>
<tr>
<td>char16 X</td>
<td>Character</td>
<td>Character getX();</td>
<td>void setX(Character x);</td>
</tr>
</tbody>
</table>
The following table lists the meta qualifiers that refine the definition of the meta constructs in the model. These qualifiers are mutually exclusive and are used to refine the actual usage of an object class or property declaration within the MOF syntax.

### TABLE 7–4  Meta Qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Scope</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association</td>
<td>class</td>
<td>Boolean</td>
<td>No affect on mapping</td>
</tr>
<tr>
<td>Indication</td>
<td>class</td>
<td>Boolean</td>
<td>Class is abstract</td>
</tr>
</tbody>
</table>

The following table lists the standard qualifiers and the effect that these qualifiers have on the mapping of a CIM object to a bean. There is no support for optional qualifiers. Javadoc API documentation is produced for each interface and class based on this mapping.

### TABLE 7–5  Standard Qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Scope</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>Class, Association, Indication</td>
<td>The class is abstract and has no effect on the Java programming language interfaces.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Any</td>
<td>The information that is provided generates Javadoc comments in the source file.</td>
</tr>
<tr>
<td>DISPLAYNAME</td>
<td>Property</td>
<td>An accessor method for the display name is created: public String displayNameForProperty();</td>
</tr>
<tr>
<td>IN</td>
<td>Parameter</td>
<td>Determines the method signature.</td>
</tr>
<tr>
<td>OUT</td>
<td>Parameter</td>
<td>Determines the method parameter signature and return values.</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>Class</td>
<td>Class or interface is final.</td>
</tr>
<tr>
<td>UNITS</td>
<td>Property, Method, Parameter</td>
<td>Another accessor method is created: public String getPropertyUnits();</td>
</tr>
</tbody>
</table>
The following table describes how MOF elements map to Java elements.

**TABLE 7–5  Standard Qualifiers  (Continued)**

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Scope</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUEMAP</td>
<td>Property, Method,</td>
<td>Beans contain generated constants for each property in a CIM class that has a CIM ValueMap or a Values qualifier. The way in which the constant name and constant value are obtained to generate these class variables depends on the data type of the property and the qualifiers that the property possesses.</td>
</tr>
<tr>
<td></td>
<td>Parameter</td>
<td>Note – The ValueMap and Values qualifiers as defined in the CIM specification have meanings contrary to what the qualifier names might imply. ValueMap defines the legal set of values for a property. Values provides translation between an integer value and a string.</td>
</tr>
<tr>
<td>VALUES</td>
<td>Property, Method,</td>
<td>Beans contain generated constants for each property in a CIM class that has a CIM ValueMap or a Values qualifier. The way in which the constant name and constant value are obtained to generate these class variables depends on the data type of the property and qualifiers that the property possesses.</td>
</tr>
<tr>
<td></td>
<td>Parameter</td>
<td>Note – The ValueMap and Values qualifiers as defined in the CIM specification have meanings contrary to what the qualifier names might imply. ValueMap defines the legal set of values for a property. Values provides translation between an integer value and a string.</td>
</tr>
<tr>
<td>VERSION</td>
<td>Class, Schema,</td>
<td>Class possesses a getClassVersion() method</td>
</tr>
<tr>
<td></td>
<td>Association,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Indication</td>
<td></td>
</tr>
</tbody>
</table>

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

**TABLE 7–6  How MOF Elements Map to Java Elements**

<table>
<thead>
<tr>
<th>MOF Element</th>
<th>Java Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description qualifier</td>
<td>Description of the class, property, or method</td>
</tr>
</tbody>
</table>
### Example of Generating JavaBeans Components

The following example shows the JavaBeans components that are produced when you use the `mofcomp` command with the `-j` option.

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

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

**EXAMPLE 7–1 Generating JavaBeans Components**

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

The content of `/usr/sadm/mof/Simple.mof` is as follows:

```
#pragma include ("CIM_Core26.mof")

class Simple_Class {
    [Key, Description ("Name of the class.")] string Name;
    [Description ("Method to print the contents of the class.")] string printClass();
};
```

The content of `/tmp/bean.txt` is as follows:

```
PACKAGE=foo.com
IMPORTS=java.lang.Exception
EXCEPTIONS=Exception
```

The content of `CIMBean.java` is as follows:
package foo.com;

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

/**
   * This Interface defines the methods that must be implemented by
   * CIMBeanImpl and its subclasses. CIMBeanImpl constitutes the base
   * class of the Java source generated by 'mofcomp -j'.
   */
public interface CIMBean {
    /**
     * This method returns the CIMBean's CIMOMHandle.
     * @return CIMOMHandle handle to the CIMOM
     */
    public CIMOMHandle getCIMOMHandle();
    /**
     * This method sets the CIMBean's CIMOMHandle to the specified value.
     * @param CIMOMHandle handle to the CIMOM
     */
    public void setCIMOMHandle(CIMOMHandle handle);
    /**
     * This method returns the CIMBean's CIMInstance.
     * @return CIMInstance handle to the CIMInstance being managed
     */
    public CIMInstance getCIMInstance();
    /**
     * This method sets the CIMBean's CIMInstance to the specified value.
     * @param CIMInstance handle to the CIMInstance being managed
     */
    public void setCIMInstance(CIMInstance instance);
    /**
     * This method makes the remote call to update the CIMInstance in the
     * CIMOM.
     */
    public void update() throws CIMException;
    /**
     * This method makes the remote call to update the specified
     * CIMProperty of the CIMInstance in the CIMOM.
     * @param String property name to update in the CIMInstance
     * @param Object property value to update in the CIMProperty
     */
    public void update(String propName, Object value) throws CIMException;
GENERATING JAVABEANS COMPONENTS (Continued)

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

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

/**
 * This method returns the CIM class's Version qualifier value or
 * '"-1" if it does not have this qualifier.
 */
public String getBeanVersion();

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

} // Interface CIMBean

The content of CIMBeanImpl.java is as follows:

```java
package foo.com;

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

/**
 * This Class implements the CIMBean Interface. It is the base Class
 * of the Java source code generated by 'mofcomp -j'.
 */
public class CIMBeanImpl implements CIMBean, Serializable {
```
private CIMInstance cimInstance = null;
private CIMOMHandle cimomHandle = null;

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

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

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

/**
* This method sets the CIMBean's CIMOMHandle to the specified value.
* *
* @param CIMOMHandle handle to the CIMOM
*/
public void setCIMOMHandle(CIMOMHandle handle) {
    this.cimomHandle = handle;
} // setCIMOMHandle

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

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

/**
 * This method makes the remote call to update the CIMInstance in the CIMOM.
 */
public void update() throws CIMException {
    cimomHandle.setInstance(getObjectPath(), cimInstance);
} // update

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

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

/**
 * This is a convenience method for use by subclasses to get the Object contained in the given CIMProperty’s CIMValue.
 * NOTE: The Object returned may be null.
 */
protected Object getProperty(String propName) {
    try {
        return (cimInstance.getProperty(propName).getValue().getValue());
    } catch (NullPointerException npe) {
    }
    return ((Object)null);
} // getProperty

/**
 * This is a convenience method for use by subclasses to get
 * the String[] equivalent to the Vector contained in the given
 * CIMProperty's CIMValue.
 * NOTE: The String[] returned may be null.
 *
 * @param String property name to get the value for
 * @param String[] property Values qualifier data
 * @param Object[] property ValueMap qualifier data
 * @return String[] container of constants for property value
 */
protected String[] getArrayProperty(String propName, String[] valueArr, Object[] valueMapArr) {
    List propList = null;
    try {
        propList = ((List)cimInstance.getProperty(propName).getValue().getValue());
    } catch (NullPointerException npe) {
    }
    if (propList != null) {
        String[] returnArr;
        returnArr = new String[propList.size()];
        ListIterator listIterator = propList.listIterator();
        int counter = 0;
        while (listIterator.hasNext()) {
            returnArr[counter] = valueArr[getArrayIndex(valueMapArr, listIterator.next())];
            counter++;
        }
        return (returnArr);
    }
    return ((String[])null);
EXAMPLE 7–1  Generating JavaBeans Components  (Continued)

    } // getArrayProperty

    /**
     * This method gets the CIMInstance referenced by the property
     * value (i.e., the object path specified) and sets it in the
     * specified Bean. This method is used by accessor methods of
     * Association properties.
     *
     * @param CIMObjectPath object path for the CIMInstance
     * @param CIMBeanImpl Bean container for CIMInstance retrieved
     */
    protected void getAssociationProperty(CIMObjectPath cop, CIMBeanImp bean) throws CIMException {
        cop.setNameSpace(""; CIMInstance ci = cimomHandle.getInstance(cop, false, true, true, (String[][])null); bean.setCIMInstance(ci); bean.setCIMOMHandle(cimomHandle);
    } // getAssociationProperty

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

    /**
     * This is a convenience method for use by subclasses to set a
     * CIMValue containing a Vector equivalent to the specified
     * String[] in the CIMProperty of the specified name.
     *
     * @param String property name to get the value for
     * @param String[] property Values qualifier data
     * @param Object[] property ValueMap qualifier data
     * @param String[] property value to set in the CIMInstance
     */
    protected void setArrayProperty(String propName, String[] valueArr, Object[] valueMapArr, String[] propValues) {
        Vector vPropValue = new Vector(propValues.length);
        for (int i = 0; i < propValues.length; i++) {
            vPropValue.addElement(valueMapArr[arrayIndex(valueArr, propValues[i])]);
        }
    } // setArrayProperty
/**
   * This method returns a string array of the Key qualified property
   * name(s) in the CIMInstance. This is needed to build the
   * CIMObjectPath for the CIMInstance if it does not contain any
   * qualifier information.
   * @return String[] array of the key qualified property names
   */
public String[] getBeanKeys() {
    return ((String[])null);
} // getBeanKeys

/**
   * This method returns the CIMObjectPath of the class's CIMInstance.
   *
   * @return CIMObjectPath object path for the CIMInstance
   */
protected CIMObjectPath getObjectPath() {
    CIMObjectPath cop = new CIMObjectPath(cimInstance.getClassName());
    Vector vKeys = cimInstance.getKeyValuePairs();
    if ((vKeys != null) && (vKeys.size() > 0)) {
        cop.setKeys(vKeys);
    } else {
        String[] keyArr = getBeanKeys();
        if (keyArr != null) {
            String keyProperty;
            for (int i = 0; i < keyArr.length; i++) {
                keyProperty = keyArr[i];
                cop.addKey(keyProperty,
                           (cimInstance.getProperty(keyProperty)).getValue());
            }
        }
    }
    return (cop);
} // getObjectPath

/**
   * This convenience method returns the index of the specified
   * object in the specified array, or '-1' if the object is not
Example 7–1  Generating JavaBeans Components  (Continued)

```java
* contained in the array.
* 
* @param Object[] Object array to find index of Object in
* @param Object Object to find index of in Object array
* @return int index of Object in Object array
*/
protected int getArrayIndex(Object[] objArr, Object obj) {
    List arrList = Arrays.asList(objArr);
    return (arrList.indexOf(obj));
}

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

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

The content of `Simple_ClassBean.java` is as follows:

```java
package foo.com;

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

/**
 * This Interface contains accessor and mutator methods for all
 * properties defined in CIM class Simple Class as well as methods
 * comparable to the invokeMethods defined for this class. This Interface
```
public interface Simple_ClassBean extends CIMBean {

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

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

/**
 * This method invokes the Simple Class.printClass() method. This
 * method is described as follows:
 * @return String return value of printClass() invocation
 * @exception Exception
 */
public String printClass() throws Exception, CIMException;
}

The content of Simple_ClassBeanImpl.java is as follows:

package foo.com;

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

/**
 * This class contains accessor and mutator methods for all properties
 * defined in the CIM class Simple Class as well as methods comparable
 * to the invokeMethods defined for this class. This class implements
 * the Simple ClassBean interface. The CIM class Simple Class is

public class Simple_ClassBeanImpl extends CIMBeanImpl implements Simple_ClassBean {
    private CIMOMHandle cimomHandle = null;
    private CIMInstance cimInstance = null;
    private final static String[] keysArr = {
            "Name"};

    /**
     * This constructor creates a Simple ClassBeanImpl class which
     * implements the Simple ClassBean interface, and encapsulates the
     * CIM class Simple Class in a bean. The CIM class Simple Class
     * is described as follows:
     * @param CIMOMHandle handle to the CIMOM
     * @param CIMInstance handle to the CIMInstance being managed
     */
    public Simple_ClassBeanImpl(CIMOMHandle handle, CIMInstance instance) {
        super(handle, instance);
        this.cimomHandle = handle;
        this.cimInstance = instance;
    } // constructor

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

    /**
     * This method returns the Simple Class.Name property value. This
     * property is described as follows:
     * @return String current Name property value
     */
    public String getName() throws Exception {
        return (String)getProperty("Name");
    } // getName
}
**EXAMPLE 7-1 Generating JavaBeans Components (Continued)**

```java
} // getName

/**
 * This method sets the Simple Class.Name property value. This
 * property is described as follows:
 * @param String new Name property value
 * @exception Exception
 */
public void setName(String name) throws Exception {
    setProperty("Name", name);
} // setName

/**
 * This method invokes the Simple Class.printClass() method. This
 * method is described as follows:
 * @return String return value of printClass() invokation
 * @exception Exception
 */
public String printClass() throws Exception, CIMException {
    Vector vInParams = new Vector();
    Vector vOutParams = new Vector();
    CIMValue cv = cimomHandle.invokeMethod(cimInstance.getObjectPath(),
        "printClass", vInParams, vOutParams);
    return (String)cv.getValue();
} // printClass
}
```

} // Class Simple_ClassBeanImpl
This chapter describes WBEM security mechanisms and the features that the CIM Object Manager (CIMOM) enforces.

This chapter includes the following information:
- “WBEM Security Mechanisms” on page 127
- “Using Sun WBEM User Manager to Set Access Control” on page 131
- “Troubleshooting Problems With WBEM Security” on page 138

**WBEM Security Mechanisms**

WBEM employs several mechanisms to ensure secure access to its data:
- **Authentication** – The process of specifying a client’s user identity to the WBEM server, and then using the client’s credentials to verify the client.
- **Role assumption** – Process that assumes that a Solaris OS role-based access control (RBAC) role identity is to be used by the WBEM server to check authorization.
- **Secure messaging** – The process of adding a secure message authenticator to each client request message. This authenticator enables the WBEM server to check the origin of the message and to determine whether that message was modified during delivery.
- **Authorization** – The process of verifying that an authenticated user or a role identity has been granted access to the data that is managed by WBEM. You use the Solaris Management Console User tool and Sun WBEM User Manager for authorization management.
- **Auditing** – The process of writing an audit record of a specific operation that was performed by the WBEM server. These records track the changes that an authenticated user makes to the management data on the WBEM server system.
- **Logging** – The writing of particular security-related events in the WBEM log. You can view the WBEM log by using the Solaris Management Console Log Viewer.

Each mechanism is described in more detail in the sections that follow.
Client Authentication

When a client application connects to a CIMOM server, the client’s user identity must be authenticated by the CIMOM on the WBEM server. The user’s WBEM client must provide a Solaris OS user identity and its associated login password. The identity and credential are used in a security authentication exchange between the client and WBEM server. This exchange is to verify that the client is a valid Solaris OS user who is allowed to log in to the WBEM server system.

If the WBEM server cannot verify the user identity and credential and the user’s identity is invalid, the WBEM server returns a CIM security exception. This exception includes the NO_SUCH_PRINCIPAL error.

If the WBEM server cannot verify the user’s identity and credential and the user’s password is invalid, the server returns a CIM security exception. This exception includes the INVALID_CREDENTIAL error.

Role Assumption

A role identity can be assumed only when a WBEM user selects the Remote Method Invocation (RMI) protocol. Role assumption is not supported by the XML over HTTP protocol.

The Solaris platform implementation of WBEM supports the ability of a client to assume the identity of a Solaris OS role. When a client assumes the identity of a Solaris OS role, the client is authenticated by the CIMOM on the WBEM server. To check RBAC authorizations, the WBEM server uses the permission that is granted to the assumed role rather than the permission that is granted to the underlying user identity.

RBAC roles are described in more detail in "Role-Based Access Control (Overview)” in System Administration Guide: Security Services.

The client must provide the Solaris OS role identity and password in addition to a Solaris OS user identity and password when the client attempts to connect.

If the WBEM server cannot verify the Solaris OS role identity, the WBEM server returns a CIM security exception that includes the NO_SUCH_ROLE error.

If the role password is invalid for the specified role identity, the WBEM server returns the INVALID_CREDENTIAL error in the CIM security exception.

If both the role identity and role password are valid, but the user is not allowed to assume the role, the WBEM server returns an exception. The CANNOT_ASSUME_ROLE error is returned in the CIM security exception.
Secure Messaging

In the CIM RMI protocol, each request from the client to the WBEM server contains a message authenticator that is constructed from the message data. A one-way digest is also created with a session key that is established during the authentication exchange.

The WBEM server verifies this message authenticator. This verification guarantees two things: The request came from the same client that was authenticated The message was not modified or replayed on its way to the server

If the message was modified, replayed, or created by a source that was not the original client, the WBEM server returns a CIM security exception. This exception contains the CHECKSUM_ERROR error. The WBEM server also writes a log message to the WBEM log.

Authorization

After connecting, the server uses the authenticated user or the role identity for all authorization checks on subsequent operations with the CIM client.

WBEM supports two types of authorization checking, using the following mechanisms:

- Access control lists (ACLs) that are maintained by the WBEM server for specific name spaces
- RBAC authorizations that are configured as part of the Solaris OS

The particular authorization checking mechanism that WBEM uses depends on how the MOF class provider is implemented. The particular authorization checking mechanism that WBEM uses for a specific MOF class operation depends on the following factors:

- The particular operation that WBEM executes
- How the MOF class provider is implemented

The classes defined in Solaris_Acl.mof implement WBEM ACL-based security. WBEM ACL-based security provides a default authorization scheme for Solaris WBEM Services. Under specific circumstances, WBEM-based security applies to a particular set of CIM operations. ACL-based security is uniquely provided by Solaris WBEM Services.

You use Sun WBEM User Manager (wbemadmin) to establish an ACL for a specific name space on the WBEM server. Sun WBEM User Manager enables you to add user, or role, names to the ACL for the name space. In addition, you can assign each user read or write permission. Sun WBEM User Manager is described in “Using Sun WBEM User Manager to Set Access Control” on page 131 and in wbemadmin(1M).

Write permission allows a user to modify the class metadata, modify instances of MOF classes in that name space, and issue an invoke method on instances. The local WBEM server root user identity is always granted write permission to all name spaces on the server. All authenticated users without an explicit ACL entry are granted read permission by default.
Operations that include the accessing of MOF class metadata, such as `getClass`, use the WBEM ACLs. These operations check the permissions that are granted to the authenticated user by the ACL for the name space that contains the MOF class. You can set an RBAC role in an ACL entry. However, the ACL entry is always checked against the user identity rather than the role identity. In other words, you can set a role name in an ACL, but the CIMOM does not check the role name at runtime.

Operations that involve MOF class instances might include the checking of either WBEM ACLs or RBAC authorizations.

You can also grant permissions to a user, or role identity, which allow that user to access and modify the instances of MOF classes, whose providers use RBAC authorizations. You grant these permissions by using the Rights tool in the Solaris Management Console User tool. The granting of permissions to a user is described in Creating or Changing a Rights Profile in System Administration Guide: Security Services.

If the instances for a MOF class are stored in the WBEM persistent datastore, the CIMOM checks the WBEM ACL for the name space that contains the MOF class. Under the following conditions, the implementation of the MOF class provider almost always uses RBAC authorization checking:

- The provider implementation for the MOF class accesses the provider’s datastore
- The provider implementation for the MOF class accesses system data in the Solaris OS

In general, if a MOF class definition contains a Provider qualifier, the provider implementation usually makes RBAC authorization checks. If the MOF class definition does not contain a Provider qualifier, the CIMOM takes the following actions:

- Stores the instances of that class in the WBEM persistent datastore
- Checks the ACL that controls access to the name space for the class to ensure that access is granted

**Auditing**

The WBEM server writes audit records for certain events during processing. For example, the WBEM server writes audit records whenever client authentication succeeds or fails, and whenever an operation that modifies user information is executed.

The WBEM server uses the underlying Solaris Basic Security Module (BSM) to write its audit records. You must enable the BSM auditing mechanism (`bsmconv`) in the Solaris OS on the WBEM server to ensure that audit information is recorded. This command is described in `bsmconv(1M)`.
Note – If you are using the Trusted Solaris software, you do not need to enable the BSM auditing mechanism.

Logging

The WBEM server writes log records to the WBEM log for particular security events. Two examples are when an authenticated session for a client is established and when an authorization fails. You can review the WBEM log in the Solaris Management Console Log Viewer, which is described in Chapter 9, “Troubleshooting.”

You can identify security-related log events by the category Security log, which is listed in the Category column. You can view only security log messages by selecting the category Security in the Log Viewer filter dialog box. Most security log messages include the user identity of the client and the name of the client host.

Using Sun WBEM User Manager to Set Access Control

Sun WBEM User Manager (wbemadmin) enables you and other privileged users to perform the following tasks:

- Add and delete authorized users
- Set access privileges for authorized users
- Manage user authentication and user access to CIM objects on a WBEM-enabled system

Note – The user for whom you specify access control must have a Solaris OS user account.

What You Can and Cannot Do With Sun WBEM User Manager

You can set access privileges for individual name spaces or for a combination of a user and a name space. When you add a user and select a name space, the user is granted read access to CIM objects in the selected name space by default.

Note – An effective way to combine user and name space access rights is to start by restricting access to a name space. Then grant individual users read, read and write, or write access to that name space.

You cannot set access rights on individual managed objects. However, you can set access rights for all managed objects in a name space as well as on a per-user basis.
If you log in as root, you can set the following types of access to CIM objects:

- **Read Only** – Allows read-only access to CIM schema objects. Users with this privilege can retrieve instances and classes, but cannot create, delete, or modify CIM objects.
- **Read/Write** – Allows full read, write, and delete access to all CIM classes, instances, and invoked methods.
- **Write** – Allows write and delete access, but not read access, to all CIM classes and instances.
- **None** – Allows no access to CIM classes and instances.

### Using Sun WBEM User Manager

This section describes how to start and use Sun WBEM User Manager.

#### How to Start Sun WBEM User Manager

1. **Become superuser.**

2. **In a command window, type the following command:**
   ```bash
   # /usr/sadm/bin/wbemadmin
   ```
   Sun WBEM User Manager starts, and a Login dialog box opens.

   **Note** – Context-help information is available in the Context Help panel when you click on the fields in the Login dialog box.

3. **Fill in the fields on the Login dialog box.**

   a. **In the User Name field, type the user name.**
      
      **Note** – You must have read access to the `root\security` name space to log in. By default, Solaris OS users have guest privileges, which grant them read access to the default name spaces. Users with read access can view but not change user privileges.

      You must log in as root or a user with write access to the `root\security` name space to grant access rights to users.

   b. **In the Password field, type the password for the user account.**

4. **Click OK.**

   The User Manager dialog box opens. The dialog box contains a list of users and their access rights to WBEM objects within the name spaces on the current host.
How to Grant Default Access Rights to a User

1. Start Sun WBEM User Manager.

2. In the Users Access portion of the dialog box, click Add. A dialog box opens that lists the available name spaces.

3. Type the name of a Solaris OS user account in the User Name field.

4. Select a name space from the listed name spaces.

5. Click OK. The user name is added to the User Manager dialog box.

6. To save changes and close the User Manager dialog box, click OK. To save changes and keep the dialog box open, click Apply. The user that you specified is granted read access to CIM objects in the name space that you selected.

How to Change Access Rights for a User

1. Start Sun WBEM User Manager.

2. Select the user whose access rights you want to change.

3. Set the user privileges. To grant the user read-only access, click the Read check box. To grant the user write access, click the Write check box.

4. To save changes and close the User Manager dialog box, click OK. To save changes and keep the dialog box open, click Apply.

How to Remove Access Rights for a User

1. Start Sun WBEM User Manager.

2. In the Users Access portion of the dialog box, select the user name for which you want to remove access rights.
Click Delete to delete the user's access rights to the name space. A confirmation dialog box opens. This dialog box prompts you to confirm your decision to delete the user's access rights.

4 To confirm, click OK.

5 To save changes and close the User Manager dialog box, click OK. To save changes and keep the dialog box open, click Apply.

How to Set Access Rights for a Name Space

1 Start Sun WBEM User Manager.

2 In the Namespace Access portion of the dialog box, click Add. A dialog box opens. The dialog box lists the available name spaces.

3 Select the name space for which you want to set access rights.

Note – By default, users have read-only access to a name space.

- To allow no access to the name space, make sure that the Read and Write check boxes are not selected.
- To allow write access, select Write.
- To allow read access, select Read.

4 To save changes and close the User Manager dialog box, click OK. To save changes and keep the dialog box open, click Apply.

How to Remove Access Rights for a Name Space

1 Start Sun WBEM User Manager.

2 In the Namespace Access portion of the dialog box, select the name space for which you want to remove access control, and then click Delete. Access control is removed from the name space, and the name space is removed from the list of name spaces on the Sun WBEM User Manager dialog box.

3 To save changes and close the User Manager dialog box, click OK. To save changes and keep the dialog box open, click Apply.
Using the Solaris WBEM SDK APIs to Set Access Control

You can use the WBEM SDK's application programming interfaces (SDK APIs) to set access control on a namespace or on a per-user basis. These security classes are stored in the root/security namespace:

- **Solaris_Acl** – Base class for Solaris OS access control lists (ACLs). This class defines the string property *capability* and sets its default value to *r* (read only).
- **Solaris_UserAcl** – Represents the access control that a user has to the CIM objects within the specified namespace.
- **Solaris_NameSpaceAcl** – Represents the access control on a name space.

You can set access control for individual users to the CIM objects within a namespace by creating an instance of the **Solaris_UserACL** class. Then use the APIs to change the access rights for that instance. Similarly, you can set access control for name spaces by first creating an instance of the **Solaris_NameSpaceACL** class. Then using APIs, such as the `createInstance` method, to set the access rights for that instance.

You can combine the use of these two classes. First, use the **Solaris_NameSpaceACL** class to restrict access to all users for the objects in a namespace. Then, use the **Solaris_UserACL** class to grant selected users access to the name space.

**Solaris_UserACL Class**

The **Solaris_UserACL** class inherits the string property *capability* with a default value *r* (read only) from the **Solaris_Acl** class.

You can set the *capability* property to any one of these values for access privileges.

<table>
<thead>
<tr>
<th>Access Right</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Read</td>
</tr>
<tr>
<td>rw</td>
<td>Read and Write</td>
</tr>
<tr>
<td>w</td>
<td>Write</td>
</tr>
<tr>
<td>none</td>
<td>No access</td>
</tr>
</tbody>
</table>

The **Solaris_UserACL** class defines the following two key properties. Only one instance of the name space and user-name ACL pair can exist in a name space.
Using the Solaris WBEM SDK APIs to Set Access Control

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>nspace</td>
<td>string</td>
<td>Identifies the name space to which this ACL applies</td>
</tr>
<tr>
<td>username</td>
<td>string</td>
<td>Identifies the user to which this ACL applies</td>
</tr>
</tbody>
</table>

### ▼ How to Set Access Control for a User

1. **Create an instance of the Solaris_UserAcl class.**
   
   For example:
   ```java
   /* Create a name space object initialized with root\security (name of name space) on the local host. */
   CIMNameSpace cns = new CIMNameSpace("", "root\security");
   // Connect to the root\security name space as root.
   cc = new CIMClient(cns, user, user_passwd);
   // Get the Solaris_UserAcl class
   cimclass = cc.getClass(new CIMObjectPath("Solaris_UserAcl"));
   // Create a new instance of the Solaris_UserAcl class
   ci = cimclass.newInstance();
   ...
   ```

2. **Set the capability property to the desired access rights.**
   
   For example:
   ```java
   /* Change the access rights (capability) to read/write for user Guest on objects in the root\molly name space. */
   ci.setProperty("capability", new CIMValue(new String("rw")));
   ci.setProperty("nspace", new CIMValue(new String("root\molly")));
   ci.setProperty("username", new CIMValue(new String("guest")));
   ...
   ```

3. **Update the instance.**
   
   For example:
   ```java
   // Pass the updated instance to the CIM Object Manager
   cc.createInstance(new CIMObjectPath(), ci);
   ...
   ```
Solaris_NamespaceAcl Class

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>nspace</td>
<td>string</td>
<td>Identifies the name space to which this access control list applies. Only one instance of the name space ACL can exist in a name space.</td>
</tr>
</tbody>
</table>

▶ How to Set Access Control for a Name Space

1. Create an instance of the Solaris_NamespaceAcl class.
   
   For example:

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

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

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

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

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

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

3. Update the instance.
   
   For example:

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

This section describes what to do in the following situations:

- A client (user) cannot be authenticated by the CIMOM on the WBEM server
- A role cannot be assumed
- An ACCESS_DENIED error occurs

If a Client (User) Cannot Be Authenticated by the CIMOM on the WBEM Server

If a client cannot be successfully authenticated by the CIMOM on the WBEM server, the WBEM server returns a CIM security exception. This exception is returned when the server attempts to establish the CIM client handle in the client application. The exception contains an error code that indicates why the authentication attempt failed.

If the WBEM server cannot verify the user identity and credential and the user’s identity is invalid, the WBEM server returns a CIM security exception. This exception includes the NO_SUCH_PRINCIPAL error. If the WBEM server cannot verify the user’s identity and credential and the user’s password is invalid for that user’s identity, the WBEM server returns a CIM security exception. This exception includes the INVALID_CREDENTIAL error.

If the WBEM server cannot verify the Solaris OS role identity, the WBEM server returns a CIM security exception that includes the NO_SUCH_ROLE error.

If the role password is invalid for the specified role identity, the WBEM server returns the INVALID_CREDENTIAL error in the CIM security exception.

If both the role identity and role password are valid but the user is not allowed to assume the role, the WBEM server returns the CANNOT_ASSUME_ROLE error in the CIM security exception.

These CIM security exceptions are described in more detail in the following table.

<table>
<thead>
<tr>
<th>Error</th>
<th>Probable Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_SUCH_PRINCIPAL</td>
<td>Specified user identity was not valid in the Solaris OS on the WBEM server.</td>
<td>Check that the user has a valid user identity. In other words, ensure that the user can log in to the Solaris OS on the WBEM server machine. You might also need to check the name service tables. This check is to determine whether the Solaris WBEM server might be using user identities from a name service configured on the server.</td>
</tr>
<tr>
<td></td>
<td>The user account for that user identity has no password.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The user account for that user identity is locked.</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Probable Cause</td>
<td>Solution</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_CREDENTIAL</td>
<td>Password for the specified user, or assumed role, is not valid for that user in the Solaris OS on the WBEM server.</td>
<td>Check that the user’s password is correct.</td>
</tr>
<tr>
<td>NO_SUCH_ROLE</td>
<td>Role identity that is used for authentication to the WBEM server is not a valid RBAC role in the Solaris OS on that server.</td>
<td>The role identity might be valid in the <code>passwd</code> table on the server, but you cannot log into the server using that identity. The Solaris software does not allow you to log in directly to role identities. You must check the <code>passwd</code> table for the role identity, and check the <code>user_attr</code> table to ensure that the role is defined as type user. Role identities in the <code>user_attr</code> table contain an attribute in the syntax <code>type=role</code>. You can also check for a valid user or valid role identity by using the Solaris Management Console User tool. You can use User Management to check for a user, and you can use Role Management to check for a role. However, when using the User tool, you must know the correct source of the tables on the CIMOM server. In other words, if the CIMOM server is using a name service such as NIS, you must access the master server for that name service.</td>
</tr>
<tr>
<td>CANNOT_ASSUME_ROLE</td>
<td>Role identity is valid, but the specified user identity in the authentication exchange is not configured to assume that role.</td>
<td>Assign users to roles by using the Administrative Role tool in the Solaris Management Console User tool collection, which is described in “How to Change the Properties of a Role” in System Administration Guide: Security Services.</td>
</tr>
</tbody>
</table>

**If Other CIM Security Exceptions Appear**

The WBEM server can return other error indications in the CIM security exception. However, these indications typically identify a system failure in the authentication exchange. The WBEM
client configuration might not be compatible with the WBEM server configuration for the security options in the authentication exchange.

If these error indications occur, check the client application CLASSPATH setting to ensure that sunwbem.jar and the extension directory are in the CLASSPATH.

If an Authorization Check Fails

If a client is not authorized to access or modify the data associated with a request to the WBEM server, that server returns a CIM security exception. This exception includes the ACCESS_DENIED error.

The ACCESS_DENIED error indicates that a request could not be completed because the user or role does not have access to the data managed by that request.

Check the security messages in the WBEM log for the failed request. For information about viewing log data, see "Viewing Log Data Through Log Viewer" on page 141. Authorization failure messages in the WBEM log specify Access denied in the Summary column. The User column lists the name of the authenticated user or the role name that was used in the check. The Source column lists the name of the provider that is making the check. Note that the provider name that is listed in this column is not the class of the provider implementation, but a user-friendly provider name.

The detailed message contains the name of the permission that was being checked, and that permission has not been granted to the user or role.

If the permission appears as namespace: right, the authorization check was using a name space ACL. The authenticated user has not been granted that permission (read or write) for that name space.

Use Sun WBEM User Manager (wbemadmin) to grant the user the appropriate permission. Sun WBEM User Manager is described in "Using Sun WBEM User Manager to Set Access Control" on page 131.

If the permission appears as solaris:application: right, the authorization check was using an RBAC authorization.

Use the Administrative Role tool in the Solaris Management Console User tool collection to grant the rights that you want to the user or role. This procedure is described in "How to Change the Properties of a Role" in System Administration Guide: Security Services.
Troubleshooting

This chapter covers the following topics:
- How to view log data
- How to read WBEM error messages
- List of error messages

Viewing Log Data Through Log Viewer

The WBEM logging service enables application developers and writers of providers to write log messages to the log files. WBEM log files to track errors, warnings, and informational messages that the management subsystem generates. For example, you might want to write out log messages under the following conditions:
- When a system is not able to access a serial port
- When a system successfully mounts a file system
- When the number of processes that are running on a system exceeds the allowed number

After you have created a log record, you can start the Solaris Management Console application and Log Viewer. A log record is automatically created when you start the Solaris Management Console software.

You can view all details of a log record in the Solaris Management Console Log Viewer.

▼ How to Start the Solaris Management Console Application and Log Viewer

1 To start the Solaris Management Console, type this command:

```
$ smc
```
In the Navigation panel, either double-click This Computer or click the expand/compress icon next to This Computer.

A tree of commands is displayed below This Computer.

Double-click System Status.

The Log Viewer icon is displayed.

Click the Log Viewer icon.

Log Viewer starts.

About WBEM Error Messages

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

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

Parts of an Error Message

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

**EXAMPLE 9-1  Parts of an Error Message**
The MOF compiler returns the following error:

```plaintext
REF_REQUIRED = Association class CIM_Docked needs at least two refs. 
Error in line 12.
```
- **REF_REQUIRED** is the *unique identifier*.
- **CIM_Docked** is a *parameter*.
- Line 12 indicates the *line number* in the .mof file in which the error occurred.

## WBEM Error Messages

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

### ABSTRACT_INSTANCE

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

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

**Solution:** Create instances for concrete classes.

### CANNOT_ASSUME_ROLE

**Description:** This error message uses two parameters:

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

**Cause:** The specified principal cannot assume the specified role.

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

### CHECKSUM_ERROR

**Description:** This error message does not use parameters.

**Cause:** The message could not be sent because the message was damaged or because the message was corrupted. The damage could have occurred accidentally in transit or by a malicious third party.
Note – This error message is displayed when the CIMOM receives an invalid checksum. A checksum is the number of bits in a packet of data that were passed over the network. This number is used by the sender and the receiver to ensure that the data has not been corrupted or intentionally modified during transit. This number also used by the sender and receiver of the information to verify that the transmission is secure.

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

Solution: Resend the message using the security features of Solaris WBEM Services. For information about using these features of Solaris WBEM Services, see Chapter 8, "Administering Security."

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 privileges and permissions necessary to complete an action.

Solution: Request privileges to complete the operation from your system administrator or the person who is responsible for your CIMOM.

CIM_ERR_ALREADY_EXISTS
Description: Instance 1: CIM_ERR_ALREADY_EXISTS

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

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

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

Description: Instance 2: CIM_ERR_ALREADY_EXISTS

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

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

Solution: In CIM Workshop, search for existing instances to see the names that are in use. Then create the instance by using a unique name.
Description: Instance 3: CIM_ERR_ALREADY_EXISTS

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

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

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

Description: 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 that you attempted to create uses the same name as an existing qualifier type for the specified property.

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

CIM_ERR_CLASS_HAS_CHILDREN

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

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

Solution: Remove the subclasses of the given class.

CIM_ERR_CLASS_HAS_INSTANCES

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

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

Solution: Remove the instances of the given class.

CIM_ERR_FAILED

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

Cause: This error message is generic, which means that this message can be displayed for many different error conditions.

Solution: The solution varies depending on the error condition.
CIM_ERR_INVALID_PARAMETER
Description: This error message uses one parameter, {0}, which is replaced by the name of the invalid parameter.
Cause: The name of the parameter or the method is invalid.
Solution: Fix the parameter.

CIM_ERR_INVALID_QUERY
Description: This error message uses two parameters:
- {0} is replaced by the invalid part of the query.
- {1} is replaced by additional information, including the actual error in the query.
Cause: The given query either has syntactical errors or semantic errors.
Solution: Fix the errors according to the exception details. In addition, make sure that the query string and query language match.

CIM_ERR_INVALID_SUPERCLASS
Description: This error message uses two parameters:
- {0} is replaced by the name of the specified subclass.
- {1} is replaced by the name of the class for which a specified subclass does not exist.
Cause: A class is specified to belong to a subclass from a superclass, but the superclass does not exist. The specified superclass might be misspelled, or a nonexistent superclass name might have been specified in place of the intended superclass name. Or, the superclass and the subclass might have been interpolated. In other words, the specified superclass may be a subclass of the subclass. In the previous example, CIM_Chassis is specified as the superclass of CIM_Container, but CIM_Chassis is a subclass of CIM_Container.
Solution: Check the spelling and the name of the superclass to ensure that the spelling 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 CIMOM is low on memory.
Solution: Delete some class definitions and static instances to free up memory.

CIM_ERR_NOT_FOUND
Description: This instance uses one parameter, {0}, which is replaced by the name of the nonexistent class.
Cause: A class is specified but that class does not exist. The specified class might be misspelled, or a nonexistent class name might have been accidentally specified in place of the intended class name.

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

Description: 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.

Description: Instance 3: CIM_ERR_NOT_FOUND

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

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

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

CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED

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

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

Solution: Provide a supported query language.

CLASS_REFERENCE

Description: This error message uses two parameters:
- {0} parameter is replaced by the name of the class that was defined to participate in a reference.
- {1} parameter is replaced by the name of the reference.

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

Solution: Add the association qualifier or remove the reference.
**INVALID_CREDENTIAL**

*Description:* This error message does not use parameters.

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

*Solution:* There are two possible solutions to this problem.
- Use the correct password.
- Make sure your CLASSPATH contains the following directory and the following file: `/usr/sadm/lib/wbem/extension:/usr/sadm/lib/wbem/sunwbem.jar`

**INVALID_DATA**

*Description:* This error message does not use parameters.

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

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

**INVALID_QUALIFIER_NAME**

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

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

*Solution:* Include the qualifier name.

**KEY_OVERRIDE**

*Description:* This error message uses two parameters:
- `{0}` is replaced by the name of the concrete class that is in an override relationship with a class that has one or more key qualifiers.
- `{1}` is replaced by the name of the concrete class that has the key qualifier.

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

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

**KEY_REQUIRED**

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

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

**METHOD OVERRIDDEN**

**Description:** This error message uses three parameters:
- `{0}` is replaced by the name of the method that is trying to override the method represented by parameter `{1}`.
- `{1}` is replaced by the name of the method that has already been overridden by the method represented by parameter `{2}`.
- `{2}` is replaced by the name of the method that has overridden parameter `{1}`.

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

**Solution:** Specify a different method to override.

**NEW_KEY**

**Description:** This error message uses two parameters:
- `{0}` is replaced by the name of the key.
- `{1}` is replaced by the name of the class that is trying to define a new key.

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

**Solution:** Do not define a new key.

**NO_CIMOM**

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

**Cause:** The CIMOM is not running on the specified host.

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

**NO_EVENT_PROVIDER**

**Description:** An event provider cannot be found.

**Cause:** The property provider class is not found.
Solution: Ensure that the CLASSPATH 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. Ensure that the CIMOM Solaris platform provider is set and the provider qualifier is correct.

**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 CLASSPATH of the CIMOM is missing one or more of the following items:
- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

**Solution:** Set the CIMOM CLASSPATH environment variable. Ensure that the CIMOM Solaris platform provider is set and the provider qualifier is correct.

**NO_METHOD_PROVIDER**

**Description:** This error message uses two parameters:
- `{0}` is replaced by the name of the class for which the method provider cannot be found.
- `{1}` is replaced by the name of the method provider class that was specified.

**Cause:** The Java class of the specified method provider is not found. This error message indicates that the CLASSPATH of the CIMOM is missing one or more of the following items:
- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

**Solution:** Set the CIMOM CLASSPATH. Ensure that the CIMOM Solaris platform provider is set and the provider qualifier is correct.

**NO_OVERRIDDEN_METHOD**

**Description:** This error message uses two parameters:
- `{0}` is replaced by the name of the method that has overridden the method represented by `{1}`.
- `{1}` is replaced by the name of the method that has been overridden.

**Cause:** The method of a subclass is trying to override the method of the superclass. However, the method that you are trying to override does not exist in the class hierarchy because the method has not been defined.
When you override a method, you override its implementation and its signature.

**Solution:** Ensure that the method exists in the superclass.

**NO.OVERRIDDEN.PROPERTY**

**Description:** This error message uses two parameters:
- `{0}` is replaced by the name of the property that has overridden `{1}`.
- `{1}` is replaced by the name of the overriding property.

**Cause:** The property of a subclass is trying to override the property of the superclass. However, the property that you are trying to override does not exist in the class hierarchy because the method has not been defined.

**Solution:** Ensure that the property exists in the superclass.

**NO.PROPERTY.PROVIDER**

**Description:** This error message uses two parameters:
- `{0}` is replaced by the name of the class for which the property provider cannot be found.
- `{1}` is replaced by the name of the property provider class that was specified.

**Cause:** The Java class of the specified property provider is not found. This error message indicates that the CLASSPATH of the CIMOM is missing one or more of the following items:
- Name of the provider class
- Parameters of the provider class
- CIM class for which the provider is defined

**Solution:** Set the CIMOM CLASSPATH. Ensure that the CIMOM is running on the host to which you are trying to connect. If the CIMOM is not running on that host, connect to a host running the CIMOM.

**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 on what attributes are required for which qualifiers, see the DMTF CIM specification at http://www.dmtf.org.
NO_SUCH_METHOD

Description: This error message uses two parameters:

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

Cause: Most likely, the method was not defined for the specified class. If the method is defined for the specified class, another method name might have been misspelled in the definition.

Solution: Define the method for the specified class. Ensure that the method name and the class name are spelled correctly.

NO_SUCH_PRINCIPAL

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

Cause: The specified user account cannot be found. The user name might have been misspelled, or the user does not have a user account.

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

NO_SUCH_QUALIFIER1

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

Cause: A new qualifier was specified but that qualifier was not defined as part of the extension schema. The qualifier must be defined as part of the CIM schema or part of an extension schema. Otherwise, the qualifier is not recognized as a valid qualifier for a property or method of a particular class.

Solution: Define the qualifier as part of the extension schema, or use a standard CIM qualifier. For information about standard CIM qualifiers and the usage of qualifiers in the CIM schema, see the DMTF CIM specification at: http://www.dmtf.org.

NO_SUCH_QUALIFIER2

Description: This error message uses two parameters:

- {0} is replaced by the name of the class, property, or method that the qualifier modifies.
- {1} is replaced by the name of the qualifier that cannot be found.

Cause: A new qualifier was specified to modify a property or method of a particular class. The qualifier was not defined as part of the extension schema. The qualifier must be defined as part of the CIM schema or part of an extension schema. Otherwise, the qualifier is not recognized as a valid qualifier for a property or method of a particular class.
Solution: Define the qualifier as part of the extension schema or use a standard CIM qualifier. For information about standard CIM qualifiers and the usage of qualifiers in the CIM schema, see the DMTF CIM specification at: http://www.dmtf.org.

NO_SUCH_ROLE
Description: This error message uses one parameter, {0}, which is replaced by the role name.
Cause: The specified role cannot be found or the specified role is not a role identity.
Solution: Make sure that the input role exists. If the role is required, contact your system administrator to set up the role.

NO_SUCH_SESSION
Description: This error message uses one parameter, {0}, which is replaced by the session identifier.
Cause: The session has been closed and is subsequently being used.
Solution: Do not close the session.

NOT_EVENT_PROVIDER
Description: This error message does not use parameters.
Cause: The provider class that was found in the class path does not implement the EventProvider interface.
Solution: Ensure that the provider is correct and register the provider implements.

NOT_HELLO
Description: This error message does not use parameters.
Cause: This error message is displayed if the data in the hello message, the first message sent to the CIMOM, is corrupted.
Solution: No action is available in response to this error message. For information about the security features of Solaris WBEM Services, see Chapter 8, “Administering Security.”

NOT_INSTANCE_PROVIDER
Description: This error message uses two parameters:

- {0} is replaced by the name of the instance for which the InstanceProvider interface is being defined.
- {1} is replaced by the name of the Java provider class that does not implement the InstanceProvider interface. The InstanceProvider interface must be implemented to enumerate all instances of the specified class.
Cause: The path to the Java provider class specified by the CLASSPATH environment variable does not implement the InstanceProvider interface.
Solution: Ensure that the provider is correct and register the provider implements.

NOT_METHOD_PROVIDER

Description: This error message uses two parameters:

- \{0\} is replaced by the name of the method for which the MethodProvider interface is being defined. The MethodProvider interface causes a specified method to be implemented and to be called in a program.
- \{1\} is replaced by the name of the Java provider class that does not implement the MethodProvider interface.

Cause: The Java provider class that was found in the class path does not implement the MethodProvider interface.

Solution: Ensure that the Java provider class that is in the class path implements the MethodProvider interface. Use the following command when you declare the provider:

```java
public Solaris implements MethodProvider.
```

NOT_PROPERTY_PROVIDER

Description: This error message uses two parameters:

- \{0\} is replaced by the name of the method for which the PropertyProvider interface is being defined. The PropertyProvider interface is required to retrieve the values of the specified property.
- \{1\} is replaced by the name of the Java provider class that does not implement the PropertyProvider interface.

Cause: The Java provider class that is in the class path does not implement the PropertyProvider interface.

Solution: Ensure that the Java provider class that is in the class path implements the PropertyProvider interface. Use the following command when you declare the provider:

```java
public Solaris implements PropertyProvider.
```

NOT_RESPONSE

Description: This error message does not use parameters.

Cause: This error message is displayed when the data in a first response message from the CIMOM is corrupted.

Solution: No action is available in response to this error message. For information about the security features of Solaris WBEM Services, see Chapter 8, “Administering Security.”

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, that property cannot be overridden again.

**Solution:** Specify a different property to override.

QUALIFIER_UNOVERRIDABLE

**Description:** This error message uses two parameters:
- {0} is replaced by the name of the qualifier with the DisableOverride flavor.
- {1} is replaced by the name of the qualifier that is disabled by {0}.

**Cause:** The qualifier that is being overridden has the DisableOverride flavor.

**Solution:** Reset the ability of the qualifier to EnableOverride or to Override=True.

REF_REQUIRED

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

**Cause:** A class was defined to participate in an association, but no references were cited. The rules of the Common Information Model specify that an association must contain one or more references.

**Solution:** Add references or remove the association qualifier.

SCOPE_ERROR

**Description:** This error message uses three parameters:
- {0} is replaced by the name of the class the specified qualifier modifies.
- {1} is replaced by the name of the specified qualifier.
- {2} is replaced by the type of attribute that the qualifier modifies.

**Cause:** A qualifier was specified in a manner that conflicts with the qualifier type definition. The scope of the [READ] qualifier is the definition that directs the [READ] qualifier to modify a property. For example, if the [READ] qualifier is specified to modify a method, the SCOPE_ERROR message is returned.
Note – The Common Information Model (CIM) specification defines the types of CIM elements that a CIM qualifier can modify. This definition of the way in which a qualifier can be used is referred to as its scope. Most qualifiers, by definition, have a scope that directs the qualifiers to modify properties or methods or both. Many qualifiers have a scope that directs the qualifiers to modify parameters, classes, associations, indications, or schemas.

Solution: Confirm the scope of the specified qualifier. Refer to the Qualifiers section of the DMTF CIM specification at http://www.dmtf.org for the standard definitions of CIM qualifiers. Use a different qualifier for the results that you want to achieve, or change your program to use the qualifier according to its CIM definition.

TYPE_ERROR

Description: This error message uses five parameters:

- {0} is replaced by the name of the specified element, such as a property, method, or qualifier.
- {1} is replaced by the name of the class to which the specified element belongs.
- {2} is replaced by the type that is defined for the element.
- {3} is replaced by the type of value that is assigned.
- {4} is replaced by the actual value that is assigned.

Cause: The value of a property parameter or method parameter and its defined type are mismatched.

Solution: Match the value of the property or method with its defined type.

UNKNOWNHOST

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

Cause: A call was made to a specified host. The specified host is unavailable or that host cannot be located. You might receive this message under any of the following conditions.

- The host name was misspelled
- The host computer was moved to a different domain
- The host name has not been registered in this domain
- The host is temporarily unavailable due to system conditions

Solution: Check the spelling of the host name. Use the ping command to ensure that the host computer is responding. Check the system conditions of the host. Ensure that the host belongs to the specified domain.
VER_ERROR

Description: This error message uses one parameter, {0}, which is replaced by the version number of the running CIMOM.

Cause: The CIMOM does not support the version of the client that is trying to connect to CIMOM.

Solution: Install the appropriate version.
Solaris Platform Schema

The Solaris platform schema and CIM schema are available by default in the CIM Object Manager. You can view the MOF files, from which the Solaris platform schema and CIM schema are compiled, in /usr/sadm/mof/. CIM schema files, which implement the Core and Common models of the Common Information Model, are denoted by the use of “CIM” in their associated file names. The Solaris platform schema files, denoted by the use of “Solaris” in their file names, provide the unique extensions that Sun Microsystems has made to the Common Information Model. The MOF files that you find in /usr/sadm/mof/ will depend on the installation cluster of the system and which packages have been installed.

Documentation of the Solaris platform providers listed in this chapter is included in the MOF file in which the provider is specified.

- “Solaris_Acl.mof File” on page 161
- “Solaris_Application.mof File” on page 161
- “Solaris_CIMOM.mof File” on page 162
- “Solaris_Core.mof File” on page 162
- “Solaris_Device.mof File” on page 163
- “Solaris_DMGT1.0.mof File” on page 164
- “Solaris_Event.mof File” on page 164
- “Solaris_Network.mof File” on page 165
- “Solaris_Performance.mof File” on page 165
- “Solaris_Project.mof File” on page 165
- “Solaris_Schema.mof File” on page 166
- “Solaris_SNMP.mof File” on page 166
- “Solaris_System.mof File” on page 166
- “Solaris_Users.mof File” on page 167
- “Solaris_VM2.0.mof File” on page 167
- “WBEMServices.mof File” on page 169
Solaris Platform Schema Files

This table provides a brief overview of the Solaris platform schema files in `/usr/sadm/mof`.

<table>
<thead>
<tr>
<th>Solaris Platform Schema File</th>
<th>What This Schema File Provides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris_Acl.mof</td>
<td>Contains the classes for WBEM access control list (ACL) based security.</td>
</tr>
<tr>
<td>Solaris_Application.mof</td>
<td>Models Solaris software packages and patches in CIM.</td>
</tr>
<tr>
<td>Solaris_CIMOM.mof</td>
<td>Contains configuration information for the CIM Object Manager.</td>
</tr>
<tr>
<td>Solaris_Core.mof</td>
<td>Contains class definition for core classes, including the computer system and statistical information.</td>
</tr>
<tr>
<td>Solaris_Device.mof</td>
<td>Enables a description of your system’s processor, serial ports, printing devices, and time settings to make your computer work with the CIM Object Manager.</td>
</tr>
<tr>
<td>Solaris_DMGT1.0.mof</td>
<td>Contains classes that represent disks, disk partitions, and other device management classes.</td>
</tr>
<tr>
<td>Solaris_Event.mof</td>
<td>Defines unique Solaris platform indication handlers. The class that is defined in this file facilitates the delivery of indications to Management clients. The protocol that is used for this delivery is the implementation of the CIM Remote Method Invocation (RMI) protocol from Sun Microsystems.</td>
</tr>
<tr>
<td>Solaris_Network.mof</td>
<td>Defines classes pertaining to network domains, IP subnets, and naming services including NIS, NIS+, LDAP, DNS, and server /etc files.</td>
</tr>
<tr>
<td>Solaris_Performance.mof</td>
<td>Defines classes that pertain to the use and performance of computing resources for each user and for each project.</td>
</tr>
<tr>
<td>Solaris_Project.mof</td>
<td>Defines classes that model the Solaris platform project database.</td>
</tr>
<tr>
<td>Solaris_Schema.mof</td>
<td>Lists all of the MOF files of the Solaris platform schema, and specifies the order in which the MOF files are read and are compiled.</td>
</tr>
<tr>
<td>Solaris_SNMP.mof</td>
<td>Contains classes used to configure the SNMP provider and its communication to SNMP agents on different systems.</td>
</tr>
</tbody>
</table>
The following sections describe the contents of each schema file in more detail.

**Solaris_Acl.mof File**

The Solaris_Acl.mof file specifies the security classes in Solaris WBEM Services. This file defines these base classes for access control lists, users, and namespaces:

- Solaris_Acl
- Solaris_NamespaceAcl
- Solaris_UserAcl

**Solaris_Application.mof File**

The Solaris_Application.mof file enables you to set up packages and patches for your applications that extend the Solaris platform schema. The Solaris_Application.mof file defines the following classes:

- Solaris_InstalledSoftwareElement
- Solaris_Package
- Solaris_Patch
- Solaris_RegistrySoftwareElement
- Solaris_SoftwareElement
- Solaris_SoftwareFeature

In addition, the Solaris_Application.mof file defines the following association classes:

- Solaris_PatchPackageDependency
- Solaris_PatchToPatchDependency
- Solaris_ProductSoftwareElementDependency
- Solaris_ProductSoftwareElements
- Solaris_ProductSoftwareFeatureDependency
Solaris Platform Schema Files

- Solaris_ProductSoftwareFeatures
- Solaris_RegistryElementDependency
- Solaris_SoftwareElementDependency
- Solaris_SoftwareElementProductDependency
- Solaris_SoftwareElementSoftwareFeatureDependency
- Solaris_SoftwareFeatureDependency
- Solaris_SoftwareFeatureParentChild
- Solaris_SoftwareFeatureProductDependency
- Solaris_SoftwareFeatureSoftwareElementDependency
- Solaris_SoftwareFeatureSoftwareElements

**Solaris_CIMOM.mof File**

The `Solaris_CIMOM.mof` file contains all the system properties used by the CIM Object Manager. The `Solaris_CIMOM.mof` file defines the following classes:

- CIM_ObjectManager
- CIM_ObjectManagerCommunicationMechanism
- CIM_WBEMCommunicationMechanism
- Solaris_CIMOM
- Solaris_ObjectManagerClientProtocolAdapter
- Solaris_ObjectManagerProtocolAdapter
- Solaris_ObjectManagerProviderProtocolAdapter
- Solaris_ProviderPath

In addition, the `Solaris_CIMOM.mof` file defines the association class `CIM_CommMechanismForManager`.

**Solaris_Core.mof File**

The `Solaris_Core.mof` file is the first of the Solaris platform schema files to be compiled after the `Solaris_Schema.mof` file. This file provides the definition of the `Solaris_ComputerSystem` class of the Solaris platform provider. The `Solaris_Core.mof` file defines the following classes:

- Solaris_ComputerSystem
- Solaris_LogRecord
- Solaris_LogService
- Solaris_Product
- Solaris_SystemDownStatisticalInformation
- Solaris_SystemUpStatisticalInformation

In addition, the `Solaris_Core.mof` file defines the following association classes:

- Solaris_ProductParentChild
Solaris_Device.mof File

The Solaris_Device.mof file defines the following classes:

- Solaris_Environment
- Solaris_EthernetAdapter
- Solaris_Keyboard
- Solaris_LogEntry
- Solaris_LogServiceProperties
- Solaris_LogServiceSetting
- Solaris_MessageLog
- Solaris_MessageLogRecord
- Solaris_MessageLogSetting
- Solaris_Printer
- Solaris_PrintJob
- Solaris_PrintQueue
- Solaris_PrintSAP
- Solaris_PrintService
- Solaris_Processor
- Solaris_SerialPort
- Solaris_SerialPortConfiguration
- Solaris_SerialPortSetting
- Solaris_SoundDevice
- Solaris_SyslogRecord
- Solaris_TimeZone

In addition, the Solaris_Device.mof file defines the following association classes:

- Solaris_CpuSysinfoPerformanceMonitor
- Solaris_CpuUtilizationPerformanceMonitor
- Solaris_CpuVminfoPerformanceMonitor
- Solaris_LogInDataFile
- Solaris_OwningPrintQueue
- Solaris_PrinterServicingQueue
- Solaris_QueueForPrintService
- Solaris_RecordInLog
- Solaris_SystemTimeZone
Solaris_DMGT1.0.mof File

The Solaris_DMGT1.0.mof file contains classes that represent disks, disk partitions, and other device management classes. The Solaris_DMGT1.0.mof file defines the following classes:

- Solaris_Disk
- Solaris_DiskDrive
- Solaris_DiskPartition
- Solaris_EXTRACapacityGroup
- Solaris_GenericController
- Solaris_IDEController
- Solaris_LogicalDisk
- Solaris_MediaPresent
- Solaris_MPXIOController
- Solaris_MPXIOGroup
- Solaris_PhysicalMedia
- Solaris_PhysicalPackage
- Solaris_SCSIController
- Solaris_StorageLibrary
- Solaris_USBController

In addition, the Solaris_DMGT1.0.mof file defines the following association classes:

- Solaris_ControllerComponent
- Solaris_ControllerLogicalIdentity
- Solaris_DiskPartitionBasedOnDisk
- Solaris_DiskPartitionBasedOnFDisk
- Solaris_IDEInterface
- Solaris_MPXIOComponent
- Solaris_MPXIOCtlrLogicalIdentity
- Solaris_MPXIOInterface
- Solaris_RealizesDiskDrive
- Solaris_RealizesDiskPartition
- Solaris_RealizesExtent
- Solaris_SCSIInterface
- Solaris_USBInterface

Solaris_Event.mof File

The Solaris_Event.mof file contains classes that deal with indication handlers that are unique to the Solaris platform. These Solaris platform indication handlers are subclasses of CIM_IndicationHandler. These subclasses include Solaris_RMI Delivery and Solaris_JAVAXRMIDelivery. The client RMI protocol uses the Solaris_JAVAXRMIDelivery handler. Solaris_Event.mof contains Solaris_RMIDelivery to ensure compatibility with previous versions of WBEM.
Solaris_Network.mof File

The Solaris_Network.mof file defines classes that pertain to network domains, IP subnets, and naming services. These naming services include NIS, NIS+, LDAP, DNS, and server /etc files. The Solaris_Network.mof file defines the following classes:

- Solaris_AdminDomain
- Solaris_DnsAdminDomain
- Solaris_IPProtocolEndpoint
- Solaris_IPSubnet
- Solaris_LdapAdminDomain
- Solaris_NisAdminDomain
- Solaris_NisplusAdminDomain
- Solaris_SystemAdminDomain

Solaris_Performance.mof File

The Solaris_Performance.mof file defines classes that pertain to computing resource metrics. These classes pertain to the use and performance of computing resources for each user and for each project. The Solaris_Performance.mof file defines the following classes:

- Solaris_ActiveProject
- Solaris_ActiveUser
- Solaris_ProcessStatisticalInformation
- Solaris_ProjectProcessAggregateStatisticalInformation
- Solaris_UserProcessAggregateStatisticalInformation

In addition, the Solaris_Performance.mof file defines the following association classes:

- Solaris_ActiveProjectProcessAggregateStatistics
- Solaris_ActiveUserProcessAggregateStatistics
- Solaris_ProcessStatistics
- Solaris_ProjectProcessStatistics
- Solaris_UserProcessStatistics

Solaris_Project.mof File

The Solaris_Project.mof file defines classes that represent the Solaris platform project database. The Solaris_Project.mof file defines the class Solaris_Project. In addition, the Solaris_Project.mof file defines the association classes Solaris_ProjectGroup and Solaris_ProjectUser.
SolarisSchema.mof File

The Solaris_Schema.mof file is the high-level container of all other MOF files that make up the Solaris platform schema. This file lists the MOF files in the order in which the files must be compiled.

The Java classes that you generate from each compilation are then sent to the CIMOM. At the CIMOM, the classes are either enacted as events or sent to the CIM Object Manager Repository for storage as objects. The following listing of the Solaris_Schema.mof file shows the Include statements in the order that is required for compilation.

```
/*
Solaris Schema
Copyright (c) 2002 Sun Microsystems, Inc. All Rights Reserved.
*/
#pragma Include ("Solaris_Core.mof")
#pragma Include ("Solaris_Application.mof")
#pragma Include ("Solaris_System.mof")
#pragma Include ("Solaris_Device.mof")
#pragma Include ("Solaris_Network.mof")
#pragma Include ("Solaris_Users.mof")
#pragma Include ("Solaris_Project.mof")
#pragma Include ("Solaris_Event.mof")
#pragma Include ("Solaris_CIMOM.mof")
#pragma Include ("Solaris_SNMP.mof")

// This must be the last include since it changes the CIM namespace
#pragma Include ("Solaris_Acl.mof")
```

The compiler parses a line of the Solaris_Schema.mof file, and compiles the file that is specified in the Include statement. This process is repeated until all included files are compiled.

Solaris_SNMP.mof File

The Solaris_SNMP.mof file defines classes that pertain to configuration information for an SNMP device. The Solaris_SNMP.mof file defines the following classes:

- Solaris_SNMPGroupConf
- Solaris_SNMPSystem
- Solaris_SNMPSystemConf

Solaris_System.mof File

The Solaris_System.mof file defines the following classes:

- Solaris_CpuSysinfo
- Solaris_CpuUtilizationInformation
- Solaris_CpuVminfo
In addition, the Solaris_System.mof file defines the following association classes:

- Solaris_EepromElementSetting
- Solaris_HostedJobScheduler
- Solaris_OwningJobScheduler
- Solaris_SystemDevice

**Solaris_Users.mof File**

The Solaris_Users.mof file defines the following classes:

- Solaris_AuthorizationAttribute
- Solaris_EmailAlias
- Solaris_ExecutionProfile
- Solaris_MailBox
- Solaris_ProfileAttribute
- Solaris_ShellSAP
- Solaris_UserAccount
- Solaris_UserGroup
- Solaris_UserHomeDirectory
- Solaris_UserTemplate

**Solaris_VM2.0.mof File**

The Solaris_VM2.0.mof file defines classes that pertain to storage devices, for example:

- State database replicas within a slice
- Range of extents within a storage extent that can be used for data
• Stripes
• Concatenated stripes
• Mirrors
• RAID Level 5 devices
• UFS logging file systems
• Spare pools
• Disk sets
• Storage volumes

The Solaris_VM2.0.mof file defines the following classes:

• Solaris_Directory
• Solaris_HSFS
• Solaris_LocalFileSystem
• Solaris_MediaPresent
• Solaris_NFS
• Solaris_UFS
• Solaris_VMConcat
• Solaris_VMDiskSet
• Solaris_VMExtent
• Solaris_VMHotSparePool
• Solaris_VMMirror
• Solaris_VM Raid5
• Solaris_VMSoftPartition
• Solaris_VMStateDatabase
• Solaris_VMStorageVolume
• Solaris_VMStripe
• Solaris_VMTrans

In addition, the Solaris_VM2.0.mof file defines the following association classes:

• Solaris_DiskIOPerformanceMonitor
• Solaris_HSFSMount
• Solaris_LocalFSResidesOnExtent
• Solaris_Mount
• Solaris_NFSExport
• Solaris_NFSMount
• Solaris_UFSMount
• Solaris_VMConcatComponent
• Solaris_VMDriveInDiskSet
• Solaris_VMExtentBasedOn
• Solaris_VMExtentInDiskSet
• Solaris_VMHostInDiskSet
• Solaris_VMHotSpareInUse
• Solaris_VMHotSpares
• Solaris_VMMirrorSubmirrors
Solaris Platform Schema Files

- Solaris_VMRaid5Component
- Solaris_VMSoftPartComponent
- Solaris_VMStatistics
- Solaris_VMStripeComponent
- Solaris_VMTransLog
- Solaris_VMTransMaster
- Solaris_VMUsesHotSparePool
- Solaris_VMVolumeBasedOn

**WBEMSNIences.mof File**

The WBEMSNIences.mof file contains classes to configure the class path of the CIM object manager and its protocol adapters, both for clients and providers. The WBEMSNIences.mof file defines the following classes:

- WBEMSNIences_CIMXMLObjectManagerClientProtocolAdapter
- WBEMSNIences_Classpath
- WBEMSNIences_ClientProtocolAdapterForManager
- WBEMSNIences_ObjectManager
- WBEMSNIences_ObjectManagerClientProtocolAdapter
- WBEMSNIences_ObjectManagerProtocolAdapter
- WBEMSNIences_ObjectManagerProviderProtocolAdapter
- WBEMSNIences_ProtocolAdapterForManager
- WBEMSNIences_ProviderProtocolAdapterForManager
- WBEMSNIences_RMIOBJECTMAManagerClientProtocolAdapter
## Index

### A
application programming interfaces (APIs)
calling methods, 60
creating a namespace, 62
creating instances, 47
deleting a class, 64
deleting instances, 47
enumerating namespaces, 55
overview, 27
programming tasks, 44
retrieving classes, 61
security, 135
setting CIM qualifiers, 68
setting instances, 50

### B
base class, creating, 63

class (Continued)
deleting, 64
retrieving, 61
security, 65, 135
classes
CIMClass, 63
creating, 63
deleting, 64

### D
default namespace, 44
Distributed Management Task Force, 21

### E
Error messages, 142
error messages, WBEM security, 138–140
examples
calling a method, 60
creating a namespace, 62
creating an instance, 47
deleting a class, 64
deleting an instance, 47
enumerating namespaces, 55
retrieving a class, 61
setting CIM qualifiers, 68
setting instances, 50
exceptions, See error messages

---

class

deleteClass, 64

---

CIM Object Manager
error messages, 142
Repository, 25, 31, 32, 33, 166
startup functions, 31
stopping, 33
CIM qualifiers, setting, 68
CIM schema, files, 159–169
class

deleteClass, 64
Index

I
instance
  creating, 46
  deleting, 47
  getting and setting, 49

J
Java
  conversion from Managed Object Format (MOF), 25
  creating instances, 47
  deleting instances, 47
  integrating Java programs with native methods, 106
  setting instances, 50
  Solaris WBEM SDK example programs, 37
  Java Native Interface, 25

L
log data, viewing, 141

M
Managed Object Format
  creating base classes, 63
  description, 26
Managed Object Format (MOF)
  See also MOF files
  conversion to Java, 25
  Solaris platform schema, 160
method
  deleteInstance, 47
  deleting a namespace, 63
  enumNameSpace, 55
  getClass, 61
  getInstance, 49
  invokeMethod, 60
method, createInstance, 65, 135
Methods, calling, 60
MOF file, security caution for compiling, 115
mof comp command, security caution, 115

N
namespace, 65
  creating, 62
  default, 45
  enumerating, 55
  setting access control, 135

P
privileges, Sun WBEM User Manager, 131
provider
  definition, 25
  interfaces, 95
  restarting the CIM Object Manager, 33
  Solaris, 25
  writing a native provider, 106
  writing native, 25

Q
qualifier
  definition, 68
  example type declaration, 68
  key, 63

R
Repository
  CIM Object Manager, 25, 31, 32, 33

S
schema
  CIM
    files, 159–169
  Solaris, 26
    files, 159–169
  Solaris platform schema, 160
  security, Sun WBEM User Manager, 131
  security namespace, 44
  single provider, 95
software components, 23
Solaris platform schema, files, 159–169
Solaris providers, 25
Solaris schema, 26
Solaris WBEM SDK, programming tasks, 44
Solaris WBEM SDK error messages, See error messages
Solaris WBEM SDK example programs, See example programs
Solaris WBEM Services, 22
Solaris WBEM Services error messages, See error messages
startup functions, 31
Sun WBEM User Manager, setting user privileges, 131

W
WBEM, definition, 21
WBEM security, error messages, 138–140
WBEM Services, Solaris, 22