Solaris WBEM Services Administrator's Guide

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

Part Number 806-1791–10
February 2000

Contents

Tables

Figures

Code Examples

Preface

The *Solaris WBEM Services Administrator's Guide* explains Common Information Model (CIM) concepts and describes how to administer Web-Based Enterprise Management (WBEM) services in the Solaris™ operating environment.

Solaris WBEM Services software makes it easier for software developers to create management applications that run on Solaris and makes the Solaris operating environment easier to manage.

Who Should Use This Book

This book is written for system administrators who administer WBEM-enabled environments.

Before You Read This Book

This book requires knowledge of the following:

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

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

- *Java™ How to Program*

  H. M. Deitel and P. J. Deitel, Prentice Hall, ISBN 0–13–263401–5

- *The Java Class Libraries, Second Edition, Volume 1*, Patrick Chan, Rosanna Lee, Douglas Kramer, Addison-Wesley, ISBN 0–201–31002–3
- *CIM Tutorial*, provided by the Distributed Management Task Force

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

- Distributed Management Task Force (DMTF)

  See this site at `www.dmtf.org` for the latest developments on CIM, information about various working groups, and contact information for extending the CIM Schema.

- Rational Software

  See this site at `www.rational.com/uml` for documentation on the Unified Modeling Language (UML) and the Rose CASE tool.

How This Book Is Organized

Chapter 1 provides an overview of Solaris WBEM Services and Web-Based Enterprise Management (WBEM).

Chapter 2 describes the CIM Object Manager and explains how to stop and start it.

Chapter 3 describes security features and how to set access rights on namespaces and users.

Chapter 4 describes the command syntax for the `mofc` command and how to compile a `.mof` file.

Chapter 5 describes the logging features.

Chapter 6 describes error messages generated by components of the Solaris WBEM Services product.

Appendix A provides information about general Common Information Model (CIM) concepts.

Appendix B describes the Solaris Schema files, Managed Object Format (MOF) files that describe managed objects in the Solaris operating environment.

Glossary is a list of words and phrases found in this book and their definitions.

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

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

Accessing Sun Documentation Online

The docs.sun.com[SM] 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`.

What Typographic Conventions Mean

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

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls –a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`

TABLE P–1 Typographic Conventions *(continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **rm** *filename.*
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide.* These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

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

TABLE P–2 Shell Prompts

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

CHAPTER 1

Overview

This chapter provides an overview of Web-Based Enterprise Management (WBEM) and Solaris WBEM Services, software that makes it easier for software developers to create management applications that run on Solaris and make the Solaris operating environment easier to manage.

This chapter covers the following topics.

- About WBEM
- About the Common Information Model
- Solaris WBEM Services Software
- Sun WBEM Software Development Kit

About WBEM

Web-Based Enterprise Management (WBEM) is an industry-wide initiative that includes standards for web-based management of systems, networks, and devices on multiple platforms. This standardization enables system administrators to manage desktops, devices, and networks.

WBEM is designed to be compatible with all major existing management protocols, including Simple Network Management Protocol (SNMP), Distributed Management Interface (DMI), and Common Management Information Protocol (CMIP).

WBEM encompasses the following standards:

- Common Information Model (CIM) – Information model for describing managed resources.
- Managed Object Format (MOF) – Language for defining CIM classes and instances.

- eXtensible Markup Language (XML) – Markup language for describing managed resources on the web.

The Distributed Management Task Force (DMTF), a group representing corporations in the computer and telecommunications industries, is leading the effort to develop management standards. The goal of the DMTF is to develop an integrated approach to managing networks across platforms and protocols, resulting in cost-effective products that interoperate as flawlessly as possible. For information about DMTF initiatives and outcomes, see the DMTF web site at `www.dmtf.org`.

About the Common Information Model

This section provides a brief introduction to basic CIM terms and concepts as they are used in the Solaris WBEM Services product. For more information on CIM, see Appendix A.

CIM is an object-oriented information model for describing managed resources, such as, disks, CPUs, and operating systems. A CIM object is a representation, or model, of a managed resource, such as a printer, disk drive, or CPU. CIM objects can be shared by any WBEM-enabled system, device, or application.

Basic CIM Elements

CIM objects with similar properties and purposes are represented as CIM classes. Properties are attributes that describe a unit of data for a class. An instance is a representation of a managed object that belongs to a particular class. Instances contain actual data. For example, `solaris_computersystem` is a CIM class that represents the Solaris operating system. The Solaris operating environment running your system is an instance of the `solaris_computersystem` class. `ResetCapability` and `InstallDate` are examples of properties of the `solaris_computersystem` class.

CIM classes are grouped into meaningful collections called schemas. A schema is a group of classes with a single owner. A class must belong to only one schema. Schemas are used for administration and class naming. All class names must be unique within a particular schema. The schema name is the determining factor in differentiating classes and properties from others that may have the same name. The naming of schema, class, and property follow this syntax:

Schemaname_classname.propertyname

The CIM Models

The Common Information Model categorizes information from general to specific. Specific information, such as a representation of the Solaris environment, extends the model. CIM consists of the following three layers of information:

- Core Model – A subset of CIM not specific to any platform.
- Common Model – Information model that visually depicts concepts, functionality, and representations of entities related to specific areas of network management, such as systems, devices, and applications.
- Extensions – Information models that support the CIM Schema and represent a very specific platform, protocol, or corporate brand.

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

The Core Model

The Core Model provides the underlying, general assumptions of the managed environment—for example, that specific, requested data must be contained in a location and distributed to requesting applications or users. These assumptions are conveyed as a set of classes and associations that conceptually form the basis of the managed environment. The Core Model is meant to introduce uniformity across schemas intended to represent specific aspects of the managed environment.

For applications developers, the Core Model provides a set of classes, associations, and properties that can be used as a starting point to describe managed systems and determine how to extend the Common Model. The Core Model establishes a conceptual framework for modeling the rest of the managed environment.

The Core Model provides classes and associations to extend specific information about systems, applications, networks, devices, and other network features through the Common Model and extensions.

The Common Model

Areas of network management depicted in the Common Model are independent of a specific technology or implementation but provide the basis for the development of management applications. This model provides a set of base classes for extension into the area of five designated technology-specific schemas: Systems, Devices, Applications, Networks, and Physical.

CIM Extensions

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

Solaris WBEM Services

Solaris WBEM Services software provides Web-Based Enterprise Management (WBEM) services on the Solaris 8 operating environment. These services make it easier for software developers to create management applications that run in the Solaris operating environment, and makes the Solaris operating environment easier to manage.

Solaris WBEM Services software provides secure access and manipulation of management data. The product includes a built-in Solaris provider that allows management applications to access information about managed resources (devices and software) in the Solaris operating environment.

The CIM Object Manager accepts connections from management applications using either RMI or XML / HTTP protocols, and provides the following services to connected clients:

- Management services, in the form of a CIM Object Manager that checks the semantics and syntax of CIM data and distributes data between applications, the CIM Repository, and managed resources.
- Security services that enable administrators to control user access to CIM information.
- Logging services that consist of classes developers can use to create applications that dynamically record event data to a log record and retrieve data from a log record. Administrators can use this data to track and determine the cause of events.
- XML services that 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, such as, creating, viewing, and deleting CIM classes and instances, querying for properties that have a specified value, enumerating (getting a list of) instances or classes in a specified class hierarchy.

Software Components

Solaris WBEM Services software consists of software components that function at three layers: Application, Management, and Provider. These components interact with the operating system and hardware layers. Figure 1–1 shows the software components and their interaction at each layer.

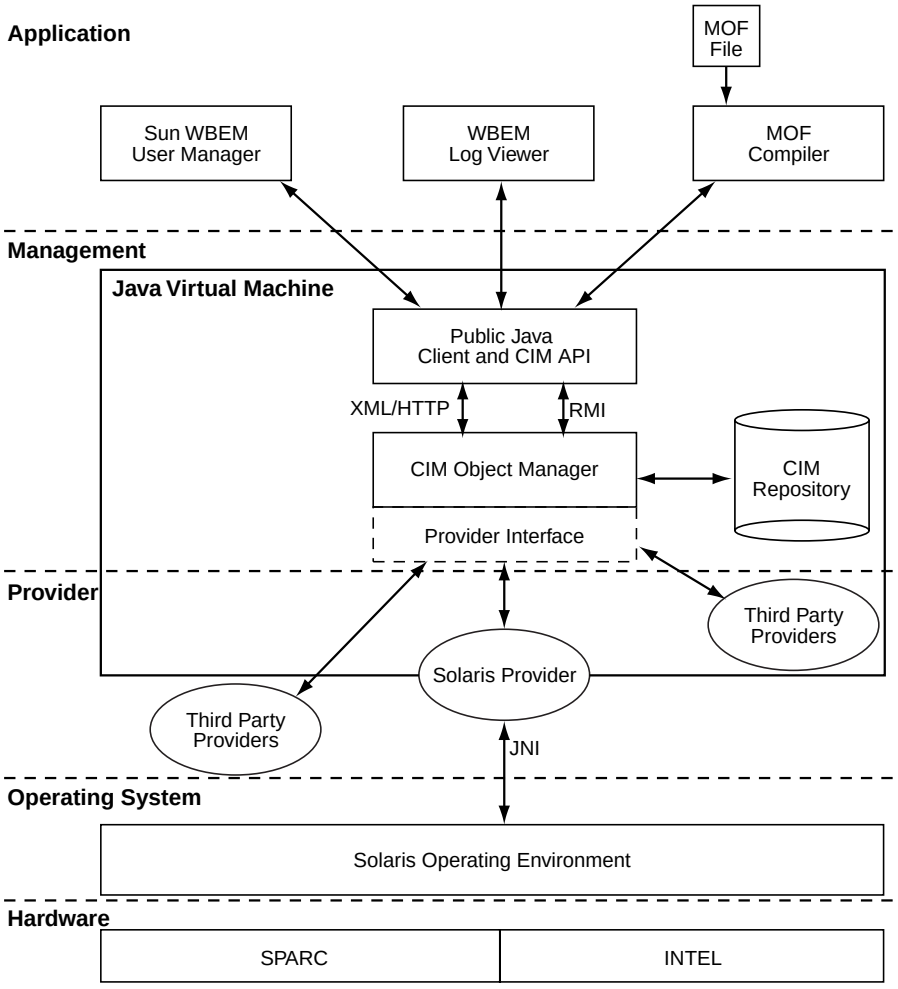

Figure 1–1 Solaris WBEM Services Architecture

- Application Layer – WBEM clients process and display data from managed resources. Solaris WBEM Services includes the following applications.

  - WBEM Log Viewer – An application that displays log files. Using the log viewer, 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.

- Sun WBEM User Manager – An application that allows administrators to add and delete authorized WBEM users and to set their access privileges to managed resources.
- Managed Object Format (MOF) Compiler – 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.

  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. For more information about MOF, see the DMTF web page at `http://www.dmtf.org`.

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

  - Client and CIM Application Programming Interfaces (APIs) – WBEM client applications use these Java interfaces to request operations, such as creating or viewing classes or instances of managed resources, from the CIM Object Manager.
  - Common Information Model (CIM) Object Manager – Software that manages CIM objects on a WBEM 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 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.
  - Provider Interface – 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 CIM Object Manager receives a request from a WBEM client for data that is not available from the CIM Object Manager Repository, it forward the request to the appropriate provider.

  - Solaris Provider – Provides the CIM Object Manager instances of managed resources in the Solaris operating environment. Providers get and set

information on managed devices. A native provider is a machine-specific program written to run on a managed device. For example, a provider that accesses data on a Solaris system will most likely include C functions to query the Solaris system. The Java Native Interface is the native programming interface for Java that is part of the JDK. By writing programs using the JNI, you ensure that your code is completely portable across all platforms. The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and assembly.

- Solaris Schema – A collection of classes that describe managed objects in the Solaris operating environment. The CIM and Solaris 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 Schema is a collection of class definitions that extend the CIM Schema and represent managed objects in a typical Solaris operating environment. Users can also use the `mofcomp` command to add CIM Schema, Solaris Schema, or other classes to the CIM Object Manager Repository after installation.

- Operating System Layer – The Solaris provider allows management applications to access information about managed resources (devices and software) in the Solaris operating environment.
- Hardware Layer – A management client can access management data on any supported Solaris platform.

Namespaces

One or more schemas can be stored in directory-like structures called namespaces. A CIM namespace is a directory-like structure that can contain other namespaces, classes, instances, and qualifier types. The names of objects within a namespace must be unique.

In Solaris WBEM Services, when WBEM client application connects to a particular namespace, all subsequent operations occur within that namespace. When connected to a namespace, the client can access the classes and instances in that namespace (if they exist) and in any namespaces contained in that namespace. For example, if you create a namespace called `child` in the `root\cimv2` namespace, you could connect to `root\cimv2` and access the classes and instances in the `root\cimv2` and `root\cimv2\child` namespaces.

An application can connect to a namespace within a namespace. This is similar to changing to a subdirectory within a directory. Once the application connects to the new namespace, all subsequent operations occur within that namespace. If you open

a new connection to `root\cimv2\child`, you can access any classes and instances in that namespace but cannot access the classes and instances in the parent namespace, `root\cimv2`.

Three namespaces are created by default during installation:

- `root` – The top-level namespace that contains other namespaces.
- `root\cimv2` – Contains the default CIM classes and instances that represent objects on your system, such as, `LogicalDisk` and `Netcard`. This is the default namespace.
- `root\security` – Contains the security classes used by the CIM Object Manager to represent access rights for users and namespaces.

Providers

When a WBEM client application accesses CIM data, the WBEM system validates the user's login information on the current host. By default, a user is granted read access to the CIM and Solaris Schema. The CIM Schema describes managed objects on your system in a standard format that all WBEM-enabled systems and applications can interpret.

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

The CIM Object Manager uses object provider application programming interfaces (APIs) to communicate with providers.

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

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

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

Interoperability with Other WBEM Systems

A WBEM client and WBEM system can run on the same system or on different systems. Multiple WBEM clients can establish connections to the same WBEM system. A typical WBEM system can serve four or five WBEM clients.

Solaris WBEM Services supports the Version 1.0 Specification for CIM Operations over HTTP. This specification uses XML to model CIM objects and messages. XML is a standard markup language for describing data on the Web. This standard extends XML markup to define CIM objects and operations. Because XML provides a standard way of describing data that can be sent across the Web, any WBEM client can access CIM data on any WBEM system that can parse XML data.

Sun WBEM Software Development Kit

The Sun WBEM Software Development Kit (SDK) contains the components required to write management applications that can communicate with any WBEM-enabled management device. Developers can also use this tool kit to write providers, programs that communicate with managed objects to access data. All management applications developed using Sun WBEM SDK run on the Java platform.

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

The Sun WBEM SDK installs and runs in any Java environment. It may be used as a stand-alone application or with Solaris WBEM Services. The Sun WBEM SDK is available from the Sun Developer Connection at `http://www.sun.com/developers/tools/sw_overview.html#foundation`.

CHAPTER 2

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 includes the following topics:

- About the CIM Object Manager
- The `init.wbem` Command
- Stopping the CIM Object Manager
- Restarting the CIM Object Manager
- Error Messages

About the CIM Object Manager

The CIM Object Manager 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 Java classes.

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

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

- Listens for RMI connections on RMI port 5987 and for XML / HTTP connections on HTTP port 80

- Sets up a connection to the CIM Object Manager Repository
- Waits for incoming requests

During normal operations, the CIM Object Manager performs these functions:

- Performs security checks to authenticate user login and authorization to access namespaces.
- Performs syntactical and semantic checking of CIM data operations to ensure that they comply with the latest CIM Specification.
- Routes requests to the appropriate provider or to the CIM Object Manager Repository. Delivers data from providers and from the CIM Object Manager Repository to WBEM client applications.
- Delivers data from providers and from the CIM Object Manager Repository to WBEM client applications.

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

The `init.wbem` Command

/etc/init.d/init.wbem start | stop

The `init.wbem` utility is run automatically during installation and each time the system is rebooted. This utility starts the CIM Boot Manager, `cimomboot`, a process that listens for connection requests from WBEM clients. When a client requests a connection, the `cimomboot` program starts the Common Information Model (CIM) Object Manager. 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.

The `init.wbem` command takes two arguments:

- `start` – Starts the CIM Boot Manager on the local host.
- `stop` – Stops the CIM Object Manager on the local host.

The `init.wbem` script is installed in the `/etc/init.d` directory. The `init.wbem` script is executed at system reboot. It is also executed by `/etc/rc2.d/S90wbem start` when init state 2 is entered, and by `/etc/rc0.d/K36wbem stop` when init state 0 is entered.

Stopping 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 root by typing the following command at the system prompt:**

```
% su
```

2. **Type the root password when you are prompted.**

3. **Stop the CIM Object Manager by typing the following command:**

```
# init.wbem stop
```

The CIM Object Manager stops.

Restarting the CIM Object Manager

Use the `init.wbem start` command to restart the CIM Object Manager. The `init.wbem` command starts the CIM Boot Manager, `cimomboot`, a process that listens for connection requests from WBEM clients.

▼ How to Restart the CIM Object Manager

1. **Become root by typing the following command at the system prompt:**

```
% su
```

2. **Type the root password when you are prompted.**

3. **Restart the CIM Boot Manager by typing the following command:**

```
# init.wbem start
```

The CIM Boot Manager starts and listens for connection requests from clients. When a client requests a connection, the CIM Boot Manager starts the CIM Object Manager.

Exception Messages

The CIM Object Manager generates exception messages to indicate incorrect MOF syntax and semantics. For an explanation of exception messages, see Chapter 6.

CHAPTER 3

Administering Security

This chapter describes the security features enforced by the CIM Object Manager, including the following topics:

- Overview
- Using the Sun WBEM User Manager to Set Access Control
- Using the APIs to Set Access Control
- CIM Exception Messages

Overview

The CIM Object Manager validates a user's login information for the machine on which the CIM Object Manager is running. A validated user is granted some form of controlled access to the entire Common Information Model (CIM) Schema. The CIM Object Manager does not provide security for system resources such as individual classes and instances. However, the CIM Object Manager does allow control of global permissions on namespace and access control on a per-user basis.

All security-related information is represented by instances of security classes located in the `root\security` namespace and must remain there permanently.

The following security features protect access to CIM objects on a WBEM-enabled system:

- Authentication - The process of verifying the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to the resources in a system.
- Authorization – The granting to a user, program, or process the right of access.

- Replay protection – A client cannot copy another client's last message sent to a CIM Object Manager. The CIM Object Manager uses the client keys to guarantee that all subsequent communication in the client-server session is with the same client that initiated the session and participated in the client-server authentication.

  The CIM Object Manager protects against a client picking up and sending another client's message to the server by validating digitally signed secret session keys. The CIM Object Manager will not accept an identical byte stream from a client without a valid secret session key.

- Digital signature – The CIM Object Manager uses Java digital signature classes to digitally sign the clients response to the server, however it does not digitally sign the server's response to a client.

Authentication

When a user logs in and enters a user name and password, the client encrypts the password and sends the encrypted password to the CIM Object Manager. When the user is authenticated, the CIM Object Manager sets up a client session. All subsequent operations occur within that secure client session.

Authorization

When a user logs in and enters a user name and password, the client encrypts the password and sends the encrypted password to the CIM Object Manager. When the user is authenticated, the CIM Object Manager sets up a client session. All subsequent operations occur within that secure client session.

Once the CIM Object Manager has authenticated the user's identity, that identity can be used to verify whether the user should be allowed to execute the application or any of its tasks. The CIM Object Manager supports capability-based authorization, which allows a privileged user to assign read and write access to specific users. These authorizations are added to existing Solaris user accounts.

Using the Sun WBEM User Manager to Set Access Control

The Sun WBEM User Manager allows privileged users to add and delete authorized users and to set their access privileges. Use this application to manage user

authentication and access to CIM objects on a WBEM-enabled system. A user must have a Solaris user account.

You can set access privileges on individual namespaces or for a user-namespace combination. When you add a user and select a namespace, by default the user is granted read access to CIM objects in the selected namespace. An effective way to combine user and namespace access rights is to first restrict access to a namespace. Then grant individual users read, read and write, or write access to that namespace.

You cannot set access rights on individual managed objects. However you can set access rights for all managed objects in a namespace as well as on a per-user basis.

If you log in to the root account, 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 and instances.
- Write – Allows write and delete, but not read access to all CIM classes and instances.
- None – Allows no access to CIM classes and instances.

▼ How to Start Sun WBEM User Manager

1. **In a command window, type the command**:

```
% /usr/sadm/bin/wbemadmin
```

   The Sun WBEM User Manager starts, and the Login dialog box opens. Context-help information is available in the Context Help panel when you click on the fields in the dialog box.

2. **In the Login dialog box, do the following**:
   - In the User Name field, type the user name.

     You must have read access to the `root\security` namespace to log in. By default, Solaris users have guest privileges, which grant them read access to the default namespaces. Users with read access can view , but cannot change, user privileges.

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

- In the Password field, type the password for the user account.

3. **Click OK.**
   The User Manager dialog box opens with a list of users and their access rights to WBEM objects within the namespaces 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 namespaces.
3. **Type the name of a Solaris user account in the User Name text entry field.**
4. **Select a namespace from the listed namespaces.**
5. **Click OK.**
   The user name is added to the User Manager dialog box.
6. **Click OK to save the changes and close the User Manager dialog box. Click Apply to save the changes and keep the dialog box open.**
   This action grants this user read access to CIM objects in the selected namespace.

▼ 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. **To grant the user read-only access, click the Read check box. To grant the user write access, click the Write check box.**
4. **Click OK to save the changes and close the User Manager dialog box. Click Apply to save the changes and keep the dialog box open.**

▼ 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.**

3. **Click Delete to delete the user's access rights to the namespace.**

   A confirmation dialog box asks you to confirm your decision to delete the user's access rights. Click OK to confirm.

4. **Click OK to save the changes and close the User Manager dialog box. Click Apply to save the changes and keep the dialog box open.**

▼ How to Set Access Rights for a Namespace

1. **Start Sun WBEM User Manager.**

2. **In the Namespace Access portion of the dialog box, click Add.**

   A dialog box opens that lists the available namespaces.

3. **Select the namespace for which you want to set access rights.**

   By default, users have read-only access to a namespace.

   - To allow no access to the namespace, make sure the Read and Write check boxes are not selected.
   - To allow write access, click the Write check box.
   - To allow read access, click the Read check box.

4. **Click OK to save the changes and close the User Manager dialog box. Click Apply to save the changes and keep the dialog box open.**

▼ How to Remove Access Rights for a Namespace

1. **Start Sun WBEM User Manager.**

2. **In the Namespace Access portion of the dialog box, select the namespace for which you want to remove access control, and then click Delete.**

   Access control is removed from the namespace, and the namespace is removed from the list of namespaces on the User Manager dialog box.

3. **Click OK to save the changes and close the User Manager dialog box. Click Apply to save the changes and keep the dialog box open.**

Using the APIs to Set Access Control

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

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

You can set access control on individual users to the CIM objects within a namespace by creating an instance of the `Solaris_UserACL` class and then using the APIs to change the access rights for that instance. Similarly, you can set access control on namespaces by creating an instance of the `Solaris_NameSpaceACL` class and then using APIs, such as the `setInstance` method, to set the access rights for that instance.

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

Note - Access Control Lists (ACL) are governed by a standard being developed by the DMTF. Although the Solaris ACL schema are currently CIM-compliant, they will need to change when the DMTF finalizes the ACL standard. Programs you write using the Solaris ACL schema classes are subject to that risk.

The `Solaris_UserAcl` Class

The `Solaris_UserAcl` class extends the `Solaris_Acl` base class, from which it inherits the string property *capability* with a default value `r` (read only).

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

Access Right	Description
`r`	Read
`rw`	Read and Write

Access Right	Description
w	Write
none	No access

The Solaris_UserAcl class defines the following two key properties. Only one instance of the namespace-username ACL pair can exist in a namespace.

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

▼ How to Set Access Control on a User

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

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

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

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

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

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

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

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

3. Update the instance. For example:

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

The `Solaris_NamespaceAcl` Class

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

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

▼ How to Set Access Control on a Namespace

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

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

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

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

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

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

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

3. Update the instance. For example:

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

Exception Messages

For a description of exception messages, see Chapter 6.

CHAPTER 4

MOF Compiler

This chapter describes the Managed Object Format (MOF) Compiler, including the following topics.

- About the MOF Compiler
- The `mofcomp` Command
- Compiling a MOF File

About the MOF Compiler

The Managed Object Format (MOF) Compiler parses a file containing MOF statements, converts the classes and instances defined in the file to Java classes, and adds the Java classes to the CIM Object Manager Repository, a central storage area for management data. The compiler loads the Java classes into the default namespace, `root\cimv2`, unless a `#pragma namespace(''`*namespace_path*`'')` statement appears in the MOF file.

The `mofcomp` command, which starts the MOF compiler, is executed before installation to compile MOF files that describe the CIM and Solaris Schemas. The CIM Schema is a collection of class definitions used to represent managed objects that occur in every management environment. The Solaris Schema is a collection of class definitions that extend the CIM Schema and represent managed objects in a typical Solaris operating environment.

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 computer representation or model of a managed resource, such as a printer, disk drive, or CPU.

Many sites store information about managed resources in MOF files. Because MOF can be converted to Java, 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.

The `mofcomp` Command

The `mofcomp` command compiles the specified MOF file into CIM classes and instances that are stored in the CIM Object Manager Repository as Java classes and passed to the CIM Object Manager.

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

/usr/sadm/bin/mofcomp [–h] [–v] [–sc] [–si] [–sq] [–version] [–c *cimom_hostname*] [–p *password*] [–u *username*] *file*

–h	List the arguments to the `mofcomp` command.
–c	Specify a system running the CIM Object Manager.
–p	Specify a password for connecting to the CIM Object Manager. Use this option for compilations that require privileged access to the CIM Object Manager. If you specify both –p and –u, you must type the password on the command line, which can pose a security risk. A more secure way to specify a password is to specify –u but not –p, so that the compiler will prompt for the password.
–sc	Run the compiler with the set class option, which updates a class if it exists and contains no instances, and returns an error if the class does not exist. If you do not specify this option, the compiler adds a CIM class to the connected namespace, and returns an error if the class already exists.
–si	Run the compiler with the set instance option, which updates an instance if it exists, and returns an error if the instance does not exist. If you do not specify this option, the compiler adds a CIM instance to the connected namespace, and returns an error if the instance already exists.
–sq	Run the compiler with the set qualifier types option, which updates a qualifier type if it exists, and returns an error if the qualifier type does not exist. If you do not

	specify this option, the compiler adds a CIM qualifier type to the connected namespace, and returns an error if the qualifier type already exists.
–u	Specify user name for connecting to the CIM Object Manager. Use this option for compilations that require privileged access to the CIM Object Manager. If you specify both –p and –u, you must type the password on the command line, which can pose a security risk. A more secure way to specify a password is to specify –u but not –p, so that the compiler will prompt for the password.
–v	Run the compiler in verbose mode, which displays compiler messages.
–version	Display the version of the MOF compiler.

The `mofcomp` command will exit with `0` upon success and a positive integer upon failure.

Compiling a MOF File

You can compile a MOF file with or without a `.mof` extension. The MOF files that describe the CIM and Solaris Schemas are located in `/usr/sadm/mof`.

▼ How to Compile a MOF File

1. **To run the MOF Compiler without parameters, type the following command:**

```
% mofcomp filename
```

For example, `mofcomp /usr/sadm/mof/Solaris_Schema1.0.mof`

The MOF file is compiled.

Examples

The following example shows the `Solaris_System1.0.mof` file, which describes managed resources in the Solaris operating environment, such as processes, operating systems, and file systems.

CODE EXAMPLE 4–1 `Solaris_System1.0` File

```
// ===================================================================
// Title:       Solaris System MOF specification 1.0
// Filename:    Solaris_System1.0.mof
// Version:     1.0
// Author:      Sun Microsystems, Inc.
// Date:        02/01/1999
// Description:
// ===================================================================

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

// ===================================================================
// Solaris_Process
// ===================================================================
[Provider("com.sun.wbem.solarisprovider.Solaris"),
 Description ("A Solaris process that is running.")
]
class Solaris_Process:CIM_Process
{
     [Description (
        "Time in user mode and kernel mode, in milliseconds."
        "If this information is not available, a value of 0 should be used."),
       Units("MilliSeconds")
     ]
   uint64 UserKernelModeTime;
     [Description (
        "A string used to identify the Parent Process. A Process ID is a "
        "kind of Process Handle."),
      MaxLen (256)
     ]
   string ParentHandle;
 };
.
.
.
// ===================================================================
// Solaris_FileSystem
// ===================================================================
  [Provider("com.sun.wbem.solarisprovider.Solaris"),
   Description ("A Solaris FileSystem.")
  ]
class Solaris_FileSystem:CIM_FileSystem
{
};
```

The following example recompiles the `Solaris_Schema1.0.mof` file with the default options, which returns an error each time the compiler attempts to add a class that already exists. This example uses verbose mode (`–v`) and specifies the root user account because that account has write access to the default namespace,

root\cimv2. By default, the CIM elements defined in the Solaris_Schema1.0.mof file will be added to the root\cimv2 namespace.

CODE EXAMPLE 4–2 Compiling a MOF File with Default Options

```
mofcomp -v -u root ../mof/Solaris_Schema1.0.mof
Starting MOF Compiler
java com.sun.wbem.compiler.mofc.CIM_Mofc -v -u root ../mof/
Solaris_Schema1.0.mof
Enter password:
Adding class Solaris_ComputerSystem
Warning at line 27 in file /usr/sadm/mof/Solaris_Core1.0.mof
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
     CIM_ERR_ALREADY_EXISTS:Element Solaris_ComputerSystem already exists.
Adding class Solaris_SerialPortSetting
Warning at line 39 in file /usr/sadm/mof/Solaris_Core1.0.mof
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
     CIM_ERR_ALREADY_EXISTS:Element Solaris_SerialPortSetting already exists.
Adding class Solaris_LogRecord
Warning at line 104 in file /usr/sadm/mof/Solaris_Core1.0.mof
- compilation proceeding ...
:
:
Semantic Error:
The following exception was thrown by setClass:
     CIM_ERR_ALREADY_EXISTS:Element Solaris_NamespaceAcl already exists.
Adding class
Warning at line 39 in file /usr/sadm/mof/Solaris_Acl1.0.mof
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
     CIM_ERR_ALREADY_EXISTS:Element Solaris_UserAcl already exists.
Compilation succeeded.
```

Security Advisory

If you run a command with the *-p* or the *-up* parameters, and you include a password, another user can run the ps command or the history command to find your password.

Note - If you run a command that requires you to provide your password, immediately change your password after running the command.

CODE EXAMPLE 4–3 Example of Unsafe Syntax

The following examples show use of the mofcomp command with the *-p* parameter:

```
mofcomp -p Log8Rif
```

and the *-up* parameters:

```
mofcomp -up molly Log8Rif
```

Change your password immediately after running the `mofcomp` command with the option to specify a password.

CHAPTER 5

Logging Events

Logging is a service that enables WBEM administrators to track events to determine how they occurred. Examples of events include recording the inaccessibility of a serial port when one is provided; the mounting of a file system which generates an error message; a system disk having reached capacity.

This chapter covers the following topics:

- About Logging
- Log Files
- Log Classes
- Using the APIs to Enable Logging
- Viewing Log Data

About Logging

The logging service records all those actions that the service provider has been programmed to return and that are completed by Solaris WBEM Services components. Informational and error content can be recorded to a log. For example, if a user disables a serial port, this information can be logged automatically by a serial port provider. Or, if a system error or other failure occurs, the administrator can check the log record to trace the cause of the occurrence.

All components, applications, and providers start logging automatically, in response to events. For example, the CIM Object Manager automatically logs events after it is installed and started.

You can set up logging for applications and providers that you develop for the WBEM environment. For information, see “Using the APIs to Enable Logging” on

page 53. You can view log data in the Log Viewer to debug the logging functionality that you develop for applications.

Log Files

When you set up an application or a provider to log events, its events are recorded in log files. All log records are stored in the path: `/usr/sadm/wbem/log`. Log files use the following naming convention:

```
wbem_log.#
```

where # is a number appended to indicate the version of the log file. A log file appended with a `.1`, such as `wbem_log.1`, is the most recently-saved version. A log file appended with a `.2` is the next oldest version. Larger file extensions, for example, `wbem_log.16`, indicate older versions of the file. Previous versions of the log file and the most recent version co-exist as an archive in `/usr/sadm/wbem/log`.

Log files are renamed with a `.1` file extension, and saved when one of the following two conditions are met:

- The current file reaches the file size limit specified by the `Solaris_LogServiceProperties` class. Default values are set in the `wbemService.properties` file.

  For information about how the properties of the `Solaris_LogServiceProperties` class control how a log file is used, see "Log File Rules" on page 50.

- The `clearLog()` method of the `Solaris_LogService` class is invoked on the current log file

  For information about the `Solaris_LogService` class and its methods, see "Solaris_LogService" on page 52.

Log File Rules

The `Solaris_LogServiceProperties` class is defined in `Solaris_Core1.0.mof`. The `Solaris_LogServiceProperties` class has properties that control the following attributes of a log file:

- Directory where the log file is written
- Name of the log file
- Size allowed for a log file before it is renamed with a `.1` file extension and saved.

- Number of log files you can have in the archive
- Ability to write log data to SysLog, the default logging system of the Solaris operating environment

To specify any of these attributes for an application that writes data to a log file, create a new instance of `Solaris_LogServiceProperties` and set the values of its associated properties. See Code Example 5–14 for detailed information about how to set property values of the new instance.

Log File Format

The logging service provides three categories of log records: application, system, and security. Log records may be informational, or may record data derived from errors or warnings. A standard set of fields are defined for the data that can be presented in logs; however, logs do not necessarily use all fields. For example, an informational log may provide a brief message describing an event. An error log may provide a more detailed message.

Some log data fields are required to identify data in the CIM Repository. These fields are properties flagged with a read-only key qualifier in the `Solaris_LogRecord` class. You cannot set the values of these fields. You can, however, set the values of any of the following fields in your log files:

- Category – Type of log record
- Severity – Severity of conditions that caused data to be written to a log file
- AppName – Name of the application from which the data was obtained
- UserName – Name of the individual who was using the application when log data was generated
- ClientMachineName – Name of the computer on which an incident occurred that generated log data.
- ServerMachineName – Name of the server on which an incident occurred that generated log data
- SummaryMessage – Brief message describing the occurrence
- DetailedMessage – Detailed message describing the occurrence
- Data – Context information that applications and providers can present to interpret a log message.

Log Classes

Logging uses two Solaris Schema classes: `Solaris_LogRecord` and `Solaris_LogService`.

Solaris_LogRecord

`Solaris_LogRecord` is defined in `Solaris_Core1.0.mof` to model an entry in a log file. When an application or provider calls the `Solaris_LogRecord` class in response to an event, the `Solaris_LogRecord` class causes all data generated by the event to be written to a log file. To see the definition of the `Solaris_LogRecord` class as part of the Solaris Provider, view the `Solaris_Core1.0.mof` file in a text editor of your choice. The `Solaris_Core1.0.mof` file is located in `/usr/sadm/mof`.

`Solaris_LogRecord` uses a vector of properties and key qualifiers to specify attributes of the events, system, user, and application or provider that generate data. Read-only qualifier values are generated transparently for use between the application and the CIM Repository. For example, the value RecordID uniquely identifies the log entry but is not displayed as part of the log format when you view generated data.

You can set the values of writable qualifier values. For example, you can set the qualifier values of properties such as `ClientMachineName` and `ServerMachineName` which identify the system on which an event occurs.

When the `SysLogFlag` property is set to true, then a detailed message of the log record is automatically sent to the `syslog` daemon on UNIX systems.

Solaris_LogService

The `Solaris_LogService` class controls the operation of the logging service and defines the ways in which log data is handled. This class has a set of methods that an application can use to distribute data about a particular event to the CIM Object Manager from the issuing application. The data becomes a trigger that generates a response from the CIM Object Manager, such as a retrieval of data from the CIM Repository.

The `Solaris_LogService` class uses the following methods:

- `clearLog` – Renames, and saves a current log file or deletes a saved log file
- `getNumRecords` – Returns the number of records in a particular log file
- `listLogFiles` – Returns a list of all log files stored in `/usr/sadm/wbem/log`

- `getCurrentLogFileName` – Returns the name of the most recent log file
- `getNumLogFiles` – Returns the number of log files stored in `/usr/sadm/wbem/log`
- `getLogFileSize` – Returns the size, in megabytes, of a particular log file
- `getSyslogSwitch` – Enables log data to be sent to SysLog, the logging service of the Solaris operating environment
- `getLogStorageName` – Returns the name of the host computer or device where log files are stored
- `getLogFileDir` – Returns the path and name of the directory where log files are stored
- `setProperties` – Enables you to set logging properties

You can view the definition of `Solaris_LogService` in the `Solaris_Core1.0.mof` file by opening the file in a text editor of your choice. The `Solaris_Core1.0.mof` file is located in `/usr/sadm/mof`.

Using the APIs to Enable Logging

Currently, you can view log file content in Log Viewer. However, you can develop your own log viewer if you prefer to view log files in a customized manner. You can use the logging application programming interfaces (APIs) to develop a log viewer. The APIs enable you to:

- Write data from an application to a log file
- Read data from a log file to your log viewer
- Set logging properties that specify how logging is handled

Writing Data to a Log File

Enabling an application to write data to a log file involves the following main tasks:

- Creating a new instance of the `Solaris_LogRecord` class
- Specifying the properties that will be written to the log file and setting values for the property qualifiers
- Setting the new instance and properties to print

▼ How to Create an Instance of `Solaris_LogRecord` to Write Data

1. **Import all necessary Java classes. The classes listed in Code Example 5–1 are the minimum classes that are required.**

CODE EXAMPLE 5–1 Importing Classes

```
import java.rmi.*;
import com.sun.wbem.client.CIMClient;
import com.sun.wbem.cim.CIMInstance;
import com.sun.wbem.cim.CIMValue;
import com.sun.wbem.cim.CIMProperty;
import com.sun.wbem.cim.CIMNameSpace;
import com.sun.wbem.cim.CIMObjectPath;
import com.sun.wbem.cim.CIMClass;
import com.sun.wbem.cim.CIMException;
import com.sun.wbem.solarisprovider.*;
import java.util.*;
```

2. **Declare the public class `CreateLog` and the following values:**
   - `CIMClient` value
   - `CIMObjectPath` value
   - `CIMNameSpace` value

CODE EXAMPLE 5–2 Declaring the `CreateLog` Class and Values

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

 if ( args.length != 3) {
     System.out.println("Usage: CreateLog host username password");
     System.exit(1);
 }

 CIMClient cc = null;
 CIMObjectPath cop = null;
 try {
     CIMNameSpace cns = new CIMNameSpace(args[0]);
     cc = new CIMClient(cns, args[1], args[2]);
```

3. Specify the vector of properties to be returned. Set values for the properties of the qualifiers.

CODE EXAMPLE 5–3 Specifying the Vector of Properties and their Values

```
Vector keys = new Vector();
CIMProperty logsvcKey;
logsvcKey = new CIMProperty("category");
logsvcKey.setValue(new CIMValue(new Integer(2)));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("severity");
logsvcKey.setValue(new CIMValue(new Integer(2)));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("AppName");
logsvcKey.setValue(new CIMValue("SomeApp"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("UserName");
logsvcKey.setValue(new CIMValue("molly"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("ClientMachineName");
logsvcKey.setValue(new CIMValue("dragonfly"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("ServerMachineName");
logsvcKey.setValue(new CIMValue("spider"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SummaryMessage");
logsvcKey.setValue(new CIMValue("brief_description"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("DetailedMessage");
logsvcKey.setValue(new CIMValue("detailed_description"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("data");
logsvcKey.setValue(new CIMValue("0xfe 0x45 0xae 0xda"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SyslogFlag");
logsvcKey.setValue(new CIMValue(new Boolean(true)));
keys.addElement(logsvcKey);
```

4. Declare the new instance of the `CIMObjectPath` for the log record.

CODE EXAMPLE 5–4 Declaring the New Instance of `CIMObjectPath`

```
CIMObjectPath logreccop = new CIMObjectPath("Solaris_LogRecord", keys);
```

5. Declare the new instance of `Solaris_LogRecord`. Set vector of properties to write to a file.

CODE EXAMPLE 5–5 Setting the Instance and Properties

```
CIMInstance ci = new CIMInstance();
  ci.setClassName("Solaris_LogRecord");
  ci.setProperties(keys);
  //System.out.println(ci.toString());
  cc.setInstance(logreccop,ci);
 }
 catch (Exception e) {
     System.out.println("Exception: "+e);
  e.printStackTrace();
 }
```

6. Close the session after data has been written to the log file.

CODE EXAMPLE 5–6 Closing the Session

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

Reading Data from a Log File

Enabling an application to read data from a log file to a log viewer involves the following tasks:

- Enumerating instances of the `Solaris_LogRecord` class
- Getting the desired instance
- Printing properties of the instance to an output device, typically a user interface

▼ How to Get an Instance of `Solaris_LogRecord` and Read Data

1. Import all the necessary Java classes. The classes in Code Example 5–7 lists the minimum required classes to be imported.

CODE EXAMPLE 5–7 Importing Classes

```
import java.rmi.*;
import com.sun.wbem.client.CIMClient;
import com.sun.wbem.cim.CIMInstance;
import com.sun.wbem.cim.CIMValue;
import com.sun.wbem.cim.CIMProperty;
import com.sun.wbem.cim.CIMNameSpace;
import com.sun.wbem.cim.CIMObjectPath;
import com.sun.wbem.cim.CIMClass;
import com.sun.wbem.cim.CIMException;
import com.sun.wbem.solarisprovider.*;
import java.util.*;
import java.util.Enumeration;
```

2. Declare the class `ReadLog`.

CODE EXAMPLE 5–8 Declaring the `ReadLog` Class

```
public class ReadLog
    {
    public static void main(String args[]) throws
    CIMException
    {
    if ( args.length != 3)
    {
    System.out.println("Usage: ReadLog host username
    password");
     System.exit(1);
```

3. Set client, objectpath, and namespace values of the `ReadLog` class.

CODE EXAMPLE 5–9 Creating a Solaris Log Record

```
}
CIMClient cc = null;
 CIMObjectPath cop = null;
try {      CIMNameSpace cns = new CIMNameSpace(args[0]);
cc = new CIMClient(cns, args[1], args[2]);
cop = new CIMObjectPath("Solaris_LogRecord");
```

4. Enumerate instances of `Solaris_LogRecord`**.**

CODE EXAMPLE 5–10 Enumerating Instances

```
Enumeration e = cc.enumInstances(cop, true);
    for (; e.hasMoreElements(); ) {
```

5. Send property values to an output device.

CODE EXAMPLE 5–11 Sending Property Values

```
System.out.println("---------------------------------");
   CIMObjectPath op = (CIMObjectPath)e.nextElement();
   CIMInstance ci = cc.getInstance(op);
   System.out.println("Record ID : " +
      (((Long)ci.getProperty("RecordID").getValue().

      getValue()).longValue()));
   System.out.println("Log filename : " +
      ((String)ci.getProperty("FileName").getValue().
      getValue()));

   int categ = (((Integer)ci.getProperty("category").
      getValue().getValue()).intValue());
   if (categ == 0)
    System.out.println("Category : Application Log");
   else if (categ == 1)
    System.out.println("Category : Security Log");
   else if (categ == 2)
    System.out.println("Category : System Log");

   int severity = (((Integer)ci.getProperty
      ("severity").getValue().getValue()).intValue());
```

(continued)

(Continuation)

```
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("AppName").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("data").getValue().getValue()));

boolean syslogflag =((Boolean)ci.getProperty("syslogflag").getValue().
   getValue()).booleanValue();
  if (syslogflag == true) {
System.out.println("Record was written to syslog as well");
} else {
System.out.println("Record was not written to
   syslog");
}
System.out.println("---------------------------------");
```

6. Return an error message to the user if an error condition occurs.

CODE EXAMPLE 5–12 Returning an Error Message

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

7. Close the session when the data has been read from the file.

CODE EXAMPLE 5–13 Closing the Session

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

Setting Logging Properties

You can create an instance of the `Solaris_LogServiceProperties` class and set property values for the instance to control how your application or provider handles logging. The following example shows how to set logging properties. Properties are stored in the `/usr/sadm/bin/wbem/wbemservices.properties` file.

CODE EXAMPLE 5–14 Setting Logging Properties

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

 if ( args.length != 3) {
     System.out.println("Usage: SetProps host username password");
     System.exit(1);
 }

 CIMClient cc = null;
 try {
     CIMNameSpace cns = new CIMNameSpace(args[0]);
     cc = new CIMClient(cns, args[1], args[2]);
```

(continued)

(Continuation)

```
   CIMObjectPath logpropcop = new CIMObjectPath("Solaris_Log
   ServiceProperties");
     Enumeration e = cc.enumInstances(logpropcop, true);
  for (; e.hasMoreElements(); ) {
   CIMObjectPath op = (CIMObjectPath)e.nextElement();
   CIMInstance ci = cc.getInstance(op);
   ci.setProperty("Directory", new CIMValue("/tmp/bar1/"));
   ci.setProperty("FileSize", new CIMValue("10"));
   ci.setProperty("NumFiles", new CIMValue("2"));
   ci.setProperty("SyslogSwitch", new CIMValue("off"));
   cc.setInstance(logpropcop,ci);
  }
 }
 catch (Exception e) {
     System.out.println("Exception: "+e);
  e.printStackTrace();
 }

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

Viewing Log Data

You can view all details of a log record in Log Viewer, an application that provides a graphical user interface for viewing recorded data. During the installation of Solaris WBEM Services, Log Viewer is installed in `/usr/sadm/bin`. To run, Log Viewer requires that an application has created a log record.

Starting Log Viewer

After you have created a log record, you can start the Log Viewer.

▼ How to Start Log Viewer

1. **Change directories to the location of Log Viewer by typing the following command:**

```
cd /usr/sadm/bin
```

2. **Start Log Viewer by typing the following command:**

```
wbemlogviewer
```

CHAPTER 6

CIM Exception Messages

This chapter describes the exception messages generated by the CIM Object Manager in the Solaris WBEM Services, including the following topics.

- How CIM Exceptions are Generated
- Parts of CIM Exceptions
- Finding Information About CIM Exceptions
- Generated CIM Exceptions

How CIM Exceptions are Generated

The CIM Object Manager generates exception messages that are used by all the clients. The MOF Compiler appends a line to the exception indicating where in a `.mof` file the error occurred. From these exceptions, client applications can generate error messages that are more meaningful to the end-user.

CIM clients can be used as XML or RMI clients. Currently, XML supports only a subset of these exceptions. If you choose to use an XML client, be aware that you may not receive all the information contained in the exception message, and that you may loose parameter information.

Parts of CIM Exceptions

CIM exception messages are made up of the following parts:

- Unique identifier – Character string that differentiates the error message from other error messages
- One or more parameters – Placeholders for the specific classes, methods, and qualifiers that are cited in the exception message

Exception Message Example

For example, the MOF Compiler may return the following exception message:

```
REF_REQUIRED
 CIM_Docked
```

where

- `REF_REQUIRED` is the unique identifier.
- `CIM_Docked` is the parameter. A parameter can be replaced with the name of any appropriate class, property, method, or qualifier.

This exception message can be turned into a more user-friendly message such as:

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

For Developers: Error Message Templates

WBEM provides exception templates for all possible error messages in the `ErrorMessages_en.properties` file. In an exception template that requires parameters, the first parameter is represented as `{0}` and the second parameter is represented as `{1}`.

The following exception template is used in the previous example:

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

Finding Information About CIM Exceptions

The following section provides a detailed explanation of each CIM exception. The exception messages are organized by unique identifiers in alphabetical order. For each exception message, the following types of information are provided, when applicable:

- Unique identifier, displayed as a heading
- Description of the parameters used in the exception message
- Example of the exception or message as it is displayed to a user, often the output of the MOF compiler or the CIM WorkShop
- Cause, or reason why the exception message was generated, and background or reference information that is helpful for understanding the error message
- Solution, including steps you can take to resolve the error are provided when available

Generated CIM Exceptions

The following section lists and describes the CIM exceptions generated by the MOF Compiler, CIM Object Manager, and WBEM client applications.

`ABSTRACT_INSTANCE`

Description

The `ABSTRACT_INSTANCE` exception has one parameter, which is the name of the abstract class.

Example

`ABSTRACT_INSTANCE` = Abstract class `ExampleClass` cannot have instances.

Cause

A client application tried to create an instance for the specified class. However, the specified class is an abstract class, and abstract classes cannot have instances.

Solution

Remove the programmed instances, as the client application cannot create such instances.

`CHECKSUM_ERROR`

Description

The `CHECKSUM_ERROR` exception has no parameters.

Example

`CHECKSUM_ERROR` = Checksum not valid.

Cause

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

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

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

Solution

Resend the message.

`CIM_ERR_ACCESS_DENIED`

Description

The `CIM_ERR_ACCESS_DENIED` exception does not have parameters.

Example

`CIM_ERR_ACCESS_DENIED` = Insufficient privileges.

Cause

This exception is displayed when a user does not have the appropriate privileges to complete an action.

Solution

See your WBEM administrator to request privileges to complete the operation.

`CIM_ERR_ALREADY_EXISTS`

Instance 1: CIM_ERR_ALREADY_EXISTS

Description

This instance of the CIM_ERR_ALREADY_EXISTS exception has one parameter which is replaced by the name of the duplicate element.

Example

CIM_ERR_ALREADY_EXISTS = Duplicate class CIM_Rack

Cause

The element you attempted to create uses the same name as an existing element.

Solution

In CIM WorkShop, search for existing elements to see the element names that are in use, then create the element using a unique element name.

Instance 2: CIM_ERR_ALREADY_EXISTS

Description

This instance of the CIM_ERR_ALREADY_EXISTS error message uses one parameter, {0}, which is replaced by the name of the duplicate instance.

Example

CIM_ERR_ALREADY_EXISTS = Duplicate instance SolarisRack

Cause

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

Solution

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

Instance 3: CIM_ERR_ALREADY_EXISTS

Description

This instance of the CIM_ERR_ALREADY_EXISTS error message uses one parameter, {0}, which is replaced by the name of the duplicate namespace.

Example

CIM_ERR_ALREADY_EXISTS = Duplicate namespace root\cimv2

Cause

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

Solution

Search for existing namespaces to see the names that are in use, then create the namespace using a unique name.

Instance 4: CIM_ERR_ALREADY_EXISTS

Description

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

Example

`CIM_ERR_ALREADY_EXISTS` = Duplicate qualifier type `Key`

Cause

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

Solution

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

`CIM_ERR_FAILED`

Description

The `CIM_ERR_FAILED` exception has one parameter which is replaced by a character string, a message that explains the error condition and its possible cause.

Example

`CIM_ERR_FAILED`=Invalid entry.

Cause

The `CIM_ERR_FAILED` exception is a generic message that can be displayed for a large number of different error conditions.

Solution

Because `CIM_ERR_FAILED` is a generic exception, many types of conditions can cause the message. The solution varies depending on the error condition.

`CIM_ERR_INVALID_PARAMETER`

Description

The `CIM_ERR_INVALID_PARAMETER` exception has one parameter which gives more information about the parameter that caused the error.

Example

`CIM_ERR_INVALID_PARAMETER` = Class `System` has no schema prefix.

Cause

An operation was performed and the parameter was invalid. For example, a class was created without providing a schema prefix in front of the class name. The

Common Information Model requires that all classes are provided with a schema prefix. For example, classes developed as part of the CIM Schema require a `CIM` prefix: `CIM_Container`. Classes developed as part of the Solaris Schema require a `Solaris` prefix: `Solaris_System`.

Solution

Provide the correct parameter. In the example above, the correct parameter would be `CIM_Container`. Find all instances of the class missing the prefix and replace them with the class name and prefix.

`CIM_ERR_INVALID_SUPERCLASS`

Description

The parameter `CIM_ERR_INVALID_SUPERCLASS` has two parameters:

- The name of the specified super class.
- The name of the sub class which caused the error.

Example

`CIM_ERR_INVALID_SUPERCLASS` = Superclass `CIM_Chassis` for class `CIM_Container` does not exist.

Cause

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

Solution

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

`CIM_ERR_NOT_FOUND`

Instance 1: CIM_ERR_NOT_FOUND

Description

The `CIM_ERR_NOT_FOUND` exception has one parameter which is replaced by the name of the non-existent element.

Example

`CIM_ERR_NOT_FOUND` = Element `Solaris_Device` does not exist.

Cause

An element has been specified for a specific operation, for example delete, but the element does not exist. The specified element may be misspelled, or a non-existent element name may have been specified accidentally in place of the intended element name.

Solution

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

Instance 2: CIM_ERR_NOT_FOUND

Description

This instance of the error message `CIM_ERR_NOT_FOUND` uses two parameters:

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

Example

`CIM_ERR_NOT_FOUND` = Instance `Solaris_EnterpriseData` does not exist for class `Solaris_ComputerSystem`.

Cause

The instance does not exist.

Solution

Create the instance.

Instance 3: CIM_ERR_NOT_FOUND

Description

This instance of the `CIM_ERR_NOT_FOUND` error message uses one parameter, `{0}`, the name of the specified namespace.

Example

`CIM_ERR_NOT_FOUND` = Namespace `verdant` does not exist.

Cause

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

Solution

Retype the name of the namespace. Ensure that typing and spelling are correct.

`CLASS_REFERENCE`

Description

The CLASS_REFERENCE exception has two parameters.

- The name of the class that contains the reference.
- The name of the reference property.

Example

CLASS_REFERENCE = Class SolarisExample1 must be declared as an association to have reference SolarisExample2

Cause

A class has been defined with a reference property. However, the class is not an association. A class can only be defined to have a reference property if it is an association.

Solution

Declare the class as an association by using the –association qualifier.

INVALID_CREDENTIAL

Description

The INVALID_CREDENTIAL exception does not have parameters.

Example

INVALID_CREDENTIAL = Invalid credentials.

Cause

This exception is displayed when an invalid password has been entered.

Solution

Retype the command and type the correct password.

INVALID_QUALIFIER_NAME

Description

The INVALID_QUALIFIER_NAME exception has one parameter which is replaced by the Managed Object Format notation that depicts an empty qualifier name.

Example

INVALID_QUALIFIER_NAME = Invalid qualifier name ‘‘ ‘‘

Cause

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

Solution

Include the qualifier name in the context of the qualifier definition.

KEY_OVERRIDE

Description

The KEY_OVERRIDE exception has two parameters:

- The overriding property.
- The overridden property.

Example

KEY_OVERRIDE = Non-key Qualifier SolarisCard cannot override key Qualifier SolarisLock.

Cause

The client has defined a class where a non-Key property is trying to override a Key property. In CIM, all concrete classes require at least one Key property, and a non-Key class cannot override a class that has a Key.

Solution

The operation is not allowed as specified in the CIM specification.

KEY_REQUIRED

Description

The KEY_REQUIRED exception has one parameter which is the name of the class that requires the key.

Example

KEY_REQUIRED = Concrete (non-abstract) class ClassName needs at least one 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

The METHOD_OVERRIDDEN command has three parameters:

- The name of the overriding method.
- The name of the overridden method.
- The name of the method that has overridden the second parameter.

Example

METHOD_OVERRIDDEN = Method Resume () cannot override Stop() which is already overridden by Start()

Cause

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

Solution

This operation is illegal.

NEW_KEY

Description

The NEW_KEY exception has two parameters.

- The name of the key.
- The name of the class that is trying to define a new key.

Example

NEW_KEY = Class CIM_PhysicalPackage cannot define new key [Key]

Cause

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

Solution

No action can be taken.

NO_CIMOM

Description

The NO_CIMOM exception has one parameter, which is the name of the host that is expected to be running the CIM Object Manager.

Example

NO_CIMOM = CIMOM molly not detected.

Cause

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

Solution

Start the CIM Object Manager by typing the command `init.wbem start` or connect to a host that is running the CIM Object Manager.

`NO_INSTANCE_PROVIDER`

Description

The `NO_INSTANCE_PROVIDER` exception has two parameters:

- The name of the class for which the instance provider cannot be found.
- The name of the instance provider class that was specified.

Example

`NO_INSTANCE_PROVIDER` = Instance provider `RPC_prop` for class `RPC_Agent` not found.

Cause

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

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

Solution

Make sure the instance provider is present in the CIM Object Manager class path.

`NO_METHOD_PROVIDER`

Description

The `NO_METHOD_PROVIDER` exception has two parameters:

- The name of the class for which the method provider cannot be found.
- The name of the method provider class that was specified.

Example

`NO_METHOD_PROVIDER` = Method provider `Start_prop` for class `RPC_Agent` not found.

Cause

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

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

Solution

Make sure the method provider is present in the CIM Object Manager class path.

`NO_OVERRIDDEN_METHOD`

Description

The `NO_OVERRIDDEN_METHOD` exception has two parameters:

- The name of the overriding method.
- The name of the overridden method.

Example

`NO_OVERRIDDEN_METHOD` = Method `Write` overridden by `Read` does not exist in class hierarchy.

Cause

The method of a subclass is trying to override a method of the superclass which does not exist anywhere in the class hierarchy.

Solution

Ensure that the method exists in the class hierarchy

`NO_OVERRIDDEN_PROPERTY`

Description

The `NO_OVERRIDDEN_PROPERTY` exception has two parameters.

- The name of the overriding property.
- The name of the overridden property.

Example

`NO_OVERRIDDEN_PROPERTY` = Property `A` overridden by `B` does not exist in class hierarchy.

Cause

The property of a subclass is trying to override the property of the superclass which does not exist anywhere in the class hierarchy.

Solution

Ensure that the property exists in the superclass hierarchy.

`NO_PROPERTY_PROVIDER`

Description

The `NO_PROPERTY_PROVIDER` error message uses two parameters:

- The name of the class for which the property provider cannot be found.
- The name of the property provider class that was specified.

Example

`NO_PROPERTY_PROVIDER` = Property provider `Write_prop` for class `RPC_Agent` not found.

Cause

The Java class of the specified property provider is not found. This error message indicates that the class specified in the path of the CIM Object Manager does not contain the class specified in the second parameter.

Solution

Set the CIM Object Manager class path.

`NO_QUALIFIER_VALUE`

Description

The `NO_QUALIFIER_VALUE` exception has two parameters:

- The name of the qualifier causing the error.
- The element to which the qualifier refers. Depending on the qualifier, the second parameter can be a class, property, method, or reference.

Example

`NO_QUALIFIER_VALUE` = Qualifier `[SOURCE]` for `Solaris_ComputerSystem` has no value.

Cause

A qualifier was specified for a property or method, but values were not included for the qualifier. For example, the qualifier `VALUES` requires a string array to be specified. If the `VALUES` qualifier is specified without the required string array, the `NO_QUALIFIER_VALUE` error message is displayed.

Solution

Specify the required parameters for the qualifier. For information about what attributes are required for which qualifiers, see the CIM Specification by the Distributed Management Task Force at the following URL: `http://dmtf.org/spec/cims.html`.

`NO_SUCH_METHOD`

Description

The `NO_SUCH_METHOD` exception has two parameters:

- The name of the specified method
- The name of the specified class

Example

`NO_SUCH_METHOD` = Method `Configure()` does not exist in class `Solaris_ComputerSystem`

Cause

Most likely, the method was not defined for the specified class. If the method is defined for the specified class, another method name may have been misspelled or typed differently in the definition.

Solution

Define the method for the specified class. Otherwise, ensure that the method name and class name were typed correctly.

`NO_SUCH_PRINCIPAL`

Description

The `NO_SUCH_PRINCIPAL` exception has one parameter which is the name of the principal, a user account.

Example

`NO_SUCH_PRINCIPAL` = Principal `molly` not found.

Cause

The specified user account cannot be found. The user name may have been mistyped upon login, or a user account has not been set up for the user.

Solution

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

`NO_SUCH_QUALIFIER1`

Description

The NO_SUCH_QUALIFIER1 exception has one parameter which is the name of the undefined qualifier.

Example

NO_SUCH_QUALIFIER1 = Qualifier [LOCAL] not found.

Cause

The qualifier does not exist in the namespace.

Solution

Define the qualifier. For information about standard CIM qualifiers and the usage of qualifiers in the CIM schema, see the CIM Specification by the Distributed Management Task Force at the following URL: http://www.dmtf.org/spec/cims.html.

NO_SUCH_QUALIFIER2

Description

The NO_SUCH_QUALIFIER2 exception has two parameters:

- The name of the class, property, or method that the qualifier modifies.
- The name of the qualifier that cannot be found.

Example

NO_SUCH_QUALIFIER2 = Qualifier [LOCAL] not found for CIM_LogicalElement

Cause

A qualifier was specified to modify a property or method of a particular class. The qualifier was not defined as part of the any schema. The qualifier is required to be defined as part of the CIM schema or an extension schema to be recognized as a valid qualifier.

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 CIM Specification by the Distributed Management Task Force at the URL, http://www.dmtf.org/spec/cims.html.

NO_SUCH_SESSION

Description

The NO_SUCH_SESSION exception has one parameter which is the session identifier.

Example

NO_SUCH_SESSION = No such session 4002.

Cause

The exception is displayed when the client session cannot be found. The CIM Object Manager removes the session for security reasons. Chapter 3.

Solution

Ensure that your CIM environment is secure.

NOT_HELLO

Description

The NOT_HELLO exception has no parameters.

Example

NOT_HELLO = Not a Hello message.

Cause

This error message is displayed if the data in the hello message—the first message sent to the CIM Object Manager—is corrupted.

Solution

Try to reconnect.

NOT_INSTANCE_PROVIDER

Description

The NOT_INSTANCE_PROVIDER exception has two parameters:

- The name of the class for which the InstanceProvider is defined.
- The name of the offending Java class.

Example

NOT_INSTANCE_PROVIDER = device_prop_provider for class Solaris_Provider does not implement InstanceProvider.

Cause

The path to the Java class specified as the provider does not implement the InstanceProvider interface.

Solution

Ensure that the Java class in the second parameter implements the InstanceProvider interface.

NOT_METHOD_PROVIDER

Description

The NOT_METHOD_PROVIDER exception has two parameters:

- The name of the method for which the MethodProvider interface is defined.
- The name of the offending Java class.

Example

NOT_METHOD_PROVIDER = Provider device_method_provider for class Solaris_Provider does not implement MethodProvider.

Cause

The path to the Java method specified in the second parameter does not implement the MethodProvider interface.

Solution

Ensure that the Java class in the second parameter implements the MethodProvider interface.

NOT_PROPERTY_PROVIDER

Description

The NOT_PROPERTY_PROVIDER exception has two parameters:

- The name of the method for which the PropertyProvider interface is defined.
- The name of the offending Java class.

Example

NOT_PROPERTY_PROVIDER = Provider device_property_provider for class Solaris_Provider does not implement PropertyProvider.

Cause

The path to the Java class in the second parameter does not implement the PropertyProvider interface.

Solution

Ensure that the Java class in the second parameter implements the PropertyProvider interface.

NOT_RESPONSE

Description

The NOT_RESPONSE exception has no parameters.

Example

NOT_RESPONSE = Not a response message.

Cause

This exception is displayed when the data in a first response message from the CIM Object Manager is corrupted.

Solution

Try to reconnect.

PROPERTY_OVERRIDDEN

Description

The PROPERTY_OVERRIDDEN exception has three parameters:

- The name of the overriding property.
- The name of the overridden property.
- The name of the method that has overridden the second parameter.

Example

PROPERTY_OVERRIDDEN = Property Volume cannot override MaxCapacity which is already overridden by RawCapacity

Cause

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

Solution

Specify a different property to override.

PS_UNAVAILABLE

Description

The PS_UNAVAILABLE exception has one parameter which is a message that describes why the persistent store became unavailable.

Example

PS_UNAVAILABLE = The persistent store is unavailable.

Cause

When the repository becomes unavailable, the first parameter gives more information on the cause.

Solution

As this exception is a general error condition, try to use the description details to see what causes the error.

`QUALIFIER_UNOVERRIDABLE`

Description

The `QUALIFIER_UNOVERRIDABLE` error message uses two parameters:

- `{0}` parameter is replaced by the name of the qualifier that is set with the `DisableOverride` flavor.
- `{1}` parameter is replaced by the name of the qualifier that is set to be disabled by `{0}`.

Example

`QUALIFIER_UNOVERRIDABLE` = `Test` cannot override qualifier `Standard` because it has `DisableOverride` flavor.

Cause

The ability of the specified qualifier to override another qualifier is disabled because the flavor of the specified qualifier has been set to `DisableOverride` or `Override=False`.

Solution

Reset the ability of the qualifier to `EnableOverride` or to `Override=True`.

`REF_REQUIRED`

Description

The `REF_REQUIRED` exception has one parameter which is the name of the association.

Example

`REF_REQUIRED` = Association class `CIM_Chassis` needs at least two references.

Cause

An association was defined without the necessary references. The rules of the Common Information Model specify that an association must contain two or more references.

Solution

Add the required references to the association in the first parameter.

`SCOPE_ERROR`

Description

The `SCOPE_ERROR` exception has three parameters:

- The name of the element the specified qualifier modifies.
- The name of the specified qualifier.
- The Meta element type of the first parameter.

Example

`SCOPE_ERROR` = Qualifier `[UNITS]` for `CIM_Container` does not have a `Property` scope.

Cause

A qualifier was specified in a manner that conflicts with the requirements of the scope definition. For example, in the CIM Specification, the `[READ]` qualifier is defined with a scope property. Hence, if you use `[READ]` to qualify a class, you will get a scope exception.

Note - The CIM Specification defines the types of CIM elements that a CIM qualifier can modify. This definition of the way in which a qualifier can be used is referred to as its scope. Most qualifiers, by definition, have a scope that directs them to modify properties or methods or both. Many qualifiers have a scope that directs them to modify parameters, classes, associations, indications, or schemas.

Solution

Confirm the scope of the specified qualifier. Refer to the section, “1.Qualifiers” of the CIM Specification by the Distributed Management Task Force at the following URL:`http://www.dmtf.org/spec/cim_spec_v20` for the standard definitions of CIM qualifiers. Use a different qualifier for the results you want to achieve, or change your program to use the qualifier according to its CIM definition.

`SIGNATURE_ERROR`

Description

The `SIGNATURE_ERROR` exception has no parameters.

Example

`SIGNATURE_ERROR` = Signature not verified

Cause

This exception is displayed when a message is corrupted either accidentally or maliciously. It differs from the checksum error in that the message has a valid checksum, but the signature cannot be verified by the public key of the client. This protection ensures that even though the session key has been compromised, only the initial client which created the session is authenticated.

Solution

No action is provided for this message, which is displayed when a session has been infringed upon by an intruder. For information about Solaris WBEM Services security features, see Chapter 3.

`TYPE_ERROR`

Description

The `TYPE_ERROR` exception has five parameters:

- The name of the specified element, such as a property, method, or qualifier.
- The name of the class to which the specified element belongs.
- The type defined for the element.
- The type of value assigned.
- The actual value assigned.

Example

`TYPE_ERROR` = Cannot convert `sint16 4` to a string for `VolumeLabel` in class `Solaris_DiskPartition`

Cause

The value of a property or method parameter and its defined type are mismatched.

Solution

Match the value of the property or method with its defined type.

`UNKNOWNHOST`

Description

The `UNKNOWNHOST` exception has one parameter which is the name of the host.

Example

`UNKNOWNHOST` = Unknown host `molly`

Cause

The client tried to connect to a host that cannot be located.

Solution

Check the spelling of the host name or contact your administrator.

`VER_ERROR`

Description

The `VER_ERROR` exception has one parameter which is the version number of the CIM Object Manager to which the client tried to connect.

Example

VER_ERROR = Unsupported version 0.

Cause

The CIM Object Manager you are trying to connect to does not support the client version.

Solution

Either upgrade the client API or upgrade the CIM Object Manager.

APPENDIX **A**

Common Information Model (CIM) Terms and Concepts

CIM Concepts

The following sections describe basic CIM terms and concepts that are essential to understanding how network entities and management functions are described and related within the context of CIM. For more detailed information about the Common Information Model and object-oriented modeling practices, including how to model your own schema, refer to the CIM Tutorial at `http://dmtf.org/spec/cim_tutorial` provided by the Distributed Management Task Force.

Object-Oriented Modeling

CIM uses the principles of Object-Oriented Modeling, a way to represent an object, entity, concept, or function that has a physical or logical existence. The goal of Object-Oriented Modeling is to set a representation of a physical entity into a framework, or model, to express the qualities and functions of the entity and its relationships with other entities. In the context of CIM, Object-Oriented Modeling is used to model hardware and software elements.

Uniform Modeling Language

Models are expressed in the form of visual representation and language. CIM conventions for rendering the model are based on the diagrammatic concepts of Uniform Modeling Language (UML). UML uses shapes to represent physical entities

and lines to represent relationships. For example, in UML, classes are represented as rectangles. Each rectangle contains the name of the class it represents. A line between two rectangles represents a relationship between the two. A line that forks to join two classes to a higher-level class represents an association.

CIM diagrams add color to the diagrams to further express relationships:

- Red lines→Associations
- Blue lines→Inheritance relationships
- Green lines→Aggregation

CIM Terms

The following terms are innate to the CIM Schema.

Schema

The terms model, schema, and framework are synonymous. Each is an abstract representation of an entity that has a physical or logical existence. In CIM, a schema is a named collection of classes used for class naming and administration. Within a schema, classes and their subclasses are represented hierarchically using the following syntax: `Schemaname_classname.propertyname`. Each class name in a schema must be unique. Solaris WBEM Services includes a Solaris Schema. It contains all classes specific to the Solaris extension to CIM.

Class and Instance

In WBEM, a class is a collection of objects that represents the most basic unit of management. For example, in Solaris WBEM Services, the three main functional classes include `CIMClass`, `CIMProperty`, and `CIMInstance`.

Abstractly, classes are used to create managed objects. Class characteristics are inherited by the child objects, or instances, that are created from a class. For example, using `CIMClass`, you can create an instance, `CIMClass (Solaris_Computer_System)`.

This instance of `CIMClass` answers the question, "What is the computer system?" The value of the instance is `Solaris_Computer_System`. All instances of the same class type are created from the same class template. In the example, the name of the computer system provides a template to create managed objects of the type `Computer_System`.

Classes can be static or dynamic. Instances of static classes are stored by the CIM Object Manager and can be retrieved from the CIM Repository when a request is made. Instances of dynamic classes—classes containing data that changes regularly, such as system usage—are created by provider applications as the data changes.

Custom Classes: Extensions to CIM

For extensions to CIM, custom classes can be developed to support managed objects that are specific to their managed environment. The CIM Object Manager API provides new classes to extend CIM for the Solaris operating environment.

Property

A property defines a characteristic of a class. For example, using the `CIMProperty` class, you can define a key as a property of a particular CIM class. Values of properties can be passed back from the CIM Object Manager as a string or as a vector for a range of properties. Each property has a unique name and only one domain—the class that owns the property. A property of a given class can be overridden by a property of its subclass.

An example of a property is the `CIMProperty`, which denotes the properties of a `CIMClass`.

Method

Like properties, methods belong to the class that owns them. A method is an action the objects of a given class are programmed to complete. For example, the method `public String getName()` returns the name of an instance as a concatenation of its keys and their values. Collectively, these actions describe the behavior of the class. Methods can belong only to the class that owns them. Within the context of a class, each method must have a unique name. A method of a given class can be overridden by a method of its subclass.

New classes inherit the definition of the method from the superclass, but not the implemented method. The definition of the method, indicated by a qualifier, serves as a placeholder in which a new implemented method can be provided. The CIM Object Manager checks for methods by starting from the lowest-level class and moving up the tree to the root class searching for a qualifier type that indicates a method.

Domain

Properties and methods are declared within a class. The class that owns the property or method is referred to as the domain of the property or method.

Qualifier and Flavor

A CIM qualifier is a modifier used to characterize CIM classes, properties, methods, and parameters. Qualifiers have unique attributes, including Name, Type, and Value, that are inherited by new classes.

Indication

An indication, an object and a type of class, is created as a result of the occurrence of an event. Indications can be arranged in a type hierarchy. Indications may have properties, methods, and triggers. Triggers are system operations, such as a change made to an existing class, or events that result in the creation of new instances of an indication.

Association

An association is a class that represents a relationship between two or more classes. Associations enable the creation of multiple relationship instances for a given class. System components can be related in many different ways, and associations provide a way of representing the relationships of these components.

Because of the way associations are defined, it is possible to establish a relationship between classes without affecting any of the related classes. The addition of an association does not affect the interface of the related classes. Only associations can have references.

Reference and Range

A reference is a type of property that defines the roles of objects involved in an association. The reference specifies the role name of the class in the context of the association. The domain of a reference is an association. The range of a reference is a character string that indicates the reference type.

Override

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

Core Model Concepts

The following sections provide descriptive information about the Core Model of CIM.

System Aspects of the Core Model

The Core Model provides classes and associations you can use to develop applications in which systems and their functions are represented as managed objects. These classes and associations embody the characteristics unique to all elements that comprise a system: physical and logical elements. Physical characteristics refer to the qualities of occupying space and conforming to the elementary laws of physics. Logical characteristics represent abstractions used to manage and coordinate aspects of the physical environment, such as system state or the capabilities of a system.

In the Core Model, logical elements can include the following.

TABLE A–1 Core Model Elements

Element Name	Description
Systems	A grouping of other logical elements. Because systems are themselves logical elements, a system can be composed of other systems.
Network Components	Classes that provide a topological view of a network.
Services and Access Points	Provide a mechanism for organizing the structures that provide access to the capabilities of a system.
Devices	An abstraction or emulation of a hardware entity, that may or may not be realized in physical hardware.

The following sections describe the classes and associations provided by the Core Model to emulate the qualities of systems.

System Classes Provided by the Core Model

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

TABLE A–2 Core Model System Classes

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

System Associations Provided by the Core Model

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

Associations can have the following types of relationships:

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

Associations express the relationship between a system and the managed elements that make up the system. Two broad types of associations are used to define the relationships between classes:

The CIM Schema defines two basic types of associations:

- Component associations, which indicate that one class is part of another
- Dependency associations, which indicate that a class cannot function or exist without another class

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

Component Associations

Component associations express the relationship between the parts of a system and the system itself. Component associations describe what elements make up a system. Abstract classes that express component associations are used to create concrete associations of this type in descendent classes. The descendent concrete associations answer the question: "What composition relationships does the component, or class, have with other components?"

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

Dependency Associations

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

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

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

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

TABLE A–3 Core Model Dependencies

Dependency Association	Description
`HostedService`	An association between a Service and the System on which the functionality resides. The cardinality of this association is one-to-many. A System may host many Services. Services are weak with respect to their hosting System. Generally speaking, a Service is hosted on the System where the LogicalDevices or SoftwareFeatures that implement the Service are located. The model does not represent Services hosted across multiple systems. This is modeled as an ApplicationSystem that acts as an aggregation point for Services that are each located on a single host.
`HostedAccessPoint`	An association between a ServiceAccessPoint (SAP) and the System on which it is provided. The cardinality of this association is one-to-many and is weak with respect to the System. Each System may host many SAPs. A feature of the model is that the access point of a service can be located on the same or a different host from the system to which the service provides access. This allows the model to depict both distributed systems (an ApplicationSystem with component Service on multiple hosts) and distributed access (a Service with access points hosted on other systems).
`ServiceSAPDependency`	An association between a Service and a ServiceAccessPoint indicating that the referenced SAP is required for the Service to provide its functionality.
`SAPSAPDependency`	An association between a SAP and another SAP indicating that the latter is required in order for the former to utilize or connect with its Service.
`ServiceAccessBySAP`	An association that identifies the access points for a Service. For example, a printer may be accessed by Netware, Apple Macintosh, or Windows ServiceAccessPoints, potentially hosted on different Systems.

Example of an Extension into the Core Model

It is possible to develop many extensions into the Core Model. One possible extension includes the addition of a Managed Element class as an abstraction of the Managed System Element class. Descendents of this Managed Element class—classes

that represent objects outside the managed system domain, such as Users or Administrators—may be added to the Core Model.

Common Model Schemas

The Common Model provides a set of base classes for the following technology-specific schemas.

Systems

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

Devices

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

The representation provided by the operating system does not have a one-to-one correspondence with the physical components of the system. For example, a modem may correspond to a discrete physical component. It may just as well be provided by a multi-function card that supports a LAN adapter as well as a modem, or the modem may be provided by an ordinary process running on the system. It is very important in using or making extensions to the model to understand this distinction between Logical Devices and Physical Components and not to get them confused.

Applications

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

A fundamental characteristic of the application model is the idea of the application life cycle. An application may be in one of four states: Deployable, Installable, Executable, and Executing. The interpretation and characteristics of the various objects used to represent applications are largely tied to the mechanisms used to transform applications from one state to another.

Networks

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

Physical

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

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

APPENDIX **B**

The Solaris Schema

During installation, the CIM Object Manager compiles MOF files that describe the CIM Schema and the Solaris Schema in the directory `/usr/sadm/mof/`. CIM Schema files, which implement the Core and Common Models of the Common Information Model, are denoted by the use of CIM in their associated file names. The Solaris Schema files, denoted by the use of Solaris in their file names, provide the implementation of the Solaris extension into the Common Information Model. This appendix describes the Solaris Schema files.

- Solaris Schema Files
- The `Solaris_Schema1.0.mof` File
- The `Solaris_Core1.0.mof` File
- The `Solaris_Application1.0.mof` File
- The `Solaris_System1.0.mof` File
- The `Solaris_Device1.0.mof` File
- The `Solaris_Acl1.0.mof` File

Solaris Schema Files

The following table provides a brief overview of the Solaris Schema files located in `/usr/sadm/mof`.

TABLE B–1 Solaris Schema Files

Solaris Schema File	Description of What it Provides
`Solaris_Schema1.0.mof`	Includes all the components of the Solaris Schema. Specifies the order in which each MOF file of the Solaris Schema runs.
`Solaris_Core1.0.mof`	Enables WBEM core features to be implemented. Enables you to set locales, qualifiers, and providers.
`Solaris_Application1.0.mof`	Models Solaris packages and patches in CIM.
`Solaris_System1.0.mof`	Models the Solaris Schema components for a system, including the operating system and processes of the system. Extends CIM Schema definitions through the definition of Solaris_Process and Solaris_OperatingSystem.
`Solaris_Device1.0.mof`	Enables a description of your system's processor to make your computer work with the CIM Object Manager.
`Solaris_Acl1.0.mof`	Sets the base class and qualifiers for user ACLs.

For more detailed information about each file, see the following sections.

The `Solaris_Schema1.0.mof` File

The `Solaris_Schema1.0.mof` file is the high-level container of all other MOF files comprised by the Solaris Schema. It lists the MOF files in the order in which they must be compiled. The Java classes generated from each compilation are then sent to the CIM Object Manager, where they are either enacted as events or sent to the CIM Repository for storage as objects. The following code example shows the `Include` statements in the order required for compilation.

```
// ===================================================================
// Title:       Solaris Master MOF 1.0
// Filename:    Solaris_Schema1.0.mof
// Version:     1.0
// Author:      Sun Microsystems, Inc.
// Date:        02/01/1999
// Description:
```

(continued)

(Continuation)

```
// ===================================================================
// ===================================================================
// Includes
// ===================================================================
#pragma Include ("usr/sadm/mof/Solaris_Core1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Application1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_System1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Device1.0.mof")
//#pragma Include ("/opt/SUNWconn/wbem/schema/Solaris_Physical1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Acl1.0.mof")
```

The compiler parses a line of the `Solaris_Schema1.0.mof` file, compiles the file specified in the `Include` statement, and then parses the next line of the `Solaris_Schema1.0.mof` file, until all included files are compiled.

The `Solaris_Core1.0.mof` File

The `Solaris_Core1.0.mof` file is the first of the Solaris Schema files to be compiled after the `Solaris_Schema 1.0.mof` file. This file provides the definitions of the `Solaris_ComputerSystem` and `Solaris_SerialPortSetting` portions of the Solaris Provider, including `LogRecord`, `Solaris_Product`, `Solaris_LogService`, and so on.

Solaris_ComputerSystem Definition

The first section of `Solaris_Core1.0.mof` sets the definition of `Solaris_ComputerSystem` as an extension of the `CIM_UnitaryComputerSystem` class.

```
[Provider("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_ComputerSystem:CIM_UnitaryComputerSystem
{
};
```

Solaris_SerialPortSetting and Logging Definitions

The `Solaris_SerialPortSetting` definition extends the `CIM_SerialPortSetting` class into the Solaris operating environment, then defines the `Solaris_LogRecord` class, which defines the types of data that can be written to WBEM system logs.

```
 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_SerialPortSetting:CIM_ElementSetting
{
        [override("Element")]
    Solaris_SerialPort REF Element;
       [override("Setting")]
    Solaris_SerialPortConfiguration REF Setting;
};

 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_LogRecord
{
```

WBEM system logs fall into three general categories: application logs, system logs, and security logs. Log records may be assigned different severities, including informational, warning, and error logs. For information about using the `Solaris_LogRecord` call to enable logging in your applications, see “Using the APIs to Enable Logging” on page 53 in Chapter 5.

All log records use a standard format that is defined in the `Solaris_LogRecord` class. The properties of `Solaris_LogRecord` indicate the types of data that are passed from an application into a log record. Some of the data passed is required by the CIM Object Manager and CIM Repository to identify the recorded data. These properties are flagged with a `[read, key]` qualifier to show that they are read-only. You can view the data but not change the values of these properties. You can change the values of properties assigned `[read,write]` qualifiers. The following list shows the properties as they are assigned to `Solaris_LogRecord` in `Solaris_Core1.0.mof`.

```
{
 [read, key]
 sint64 RecordID;

 [read, key]
 sint32 RecordHashCode;

 [read, key]
 string Filename;

 [read]
```

(continued)

(Continuation)

```
 datetime RecordDate;

 [read, write]
 sint32 category;

 [read, write]
 sint32 severity;

 [read, write]
 string AppName;

 [read, write]
 string UserName;

 [read, write]
 string ClientMachineName;

 [read, write]
 string ServerMachineName;

 [read, write]
 string SummaryMessage;

 [read, write]
 string DetailedMessage;
 [read, write]
 string data;

 [read, write]
 boolean SyslogFlag;
};
```

After properties are defined for `Solaris_LogRecord`, the `Solaris_LogService` class is defined as an extension of `CIM_Service`. `Solaris_LogService` controls the operation of the logging service.

```
 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_LogService : CIM_Service
{
```

Each of the functions specified by the `Solaris_LogService` class defines how data is handled in a log file. For example, the `clearLog` function specifies that all data is deleted from the log file and the log file is refreshed to accept new data.

```
{
 sint32 clearLog([IN] string fileName);

 sint64 getNumRecords([IN] string fileName, [OUT] sint64 numRec);

 sint32 listLogFiles([OUT]string logFiles[]);

 sint32 getCurrentLogFileName([OUT] string fileName);

 sint32 getNumLogFiles([OUT] sint32 numFiles);

 sint64 getLogFileSize([OUT] sint64 fileSize);

 sint32 getSyslogSwitch([OUT] string switch);

 sint32 getLogStorageName([OUT] string fileName);

 sint32 getLogFileDir([OUT] string dirName);

 sint32 setProperties([IN] string data[]);
};
```

The `Solaris_LogServiceProperties` class controls the following characteristics of a log file:

- Directory where the log file is written
- Name of the log file
- Date the log file was created
- Size allowed for a log file before it is renamed and saved
- Number of log files you can have in the archive
- Ability to write log data to SysLog, the default logging system of the Solaris operating environment

The `Solaris_Application1.0.mof` File

The `Solaris_Application1.0.mof` file provides the ability to set up packages and patches for your applications that extend the Solaris Schema.

Packages

The `Solaris_Application1.0.mof` file contains classes that represent standard Solaris packages. These packages can be individually installed in and removed from the Solaris operating environment.

The following table lists the attributes you can set for your application packages, shows the field in which the package attribute is displayed when you run the `pkginfo` command, and describes the package attribute.

TABLE B–2 Package Information You Can Provide

Package Attribute	`pkginfo` field	Description
Name	`PKGINST`	The name you assign to your package. Names typically take the form of 3-4 uppercase characters indicating the vendor, and up to 5 lowercase characters to uniquely identify the package.
Description	`DESC`	A brief description of the package in the form of a character string.
Caption	`NAME`	An additional brief description of the package in the form of a character string.
Category	`CATEGORY`	Type of information contained in the package, for example, if the package contains video, graphic, or Java applications. The category is formed by a free-form string, that usually contains the term, *system* or *application*. The category string can be comprised of multiple terms separated by commas. Possible values include: *ALE, graphics, java, video, JFP, SyMON*.

TABLE B–2 Package Information You Can Provide *(continued)*

Architecture	ARCH	System architecture to which the package applies. The Architecture attribute can be a string or an enumeration. Use `all` to specify a generic text package such as a package that consists of man pages `sparc` or `i386` to represent a binary (represented by its processor type) `sparc.sun4u` to specify a kernal (represented as a subclass of its processor type).
Base Directory	BASEDIR	Valid UNIX path that indicates the top level directory where the package was installed.
Manufacturer	VENDOR	Manufacturer of the product.
Build Number	PSTAMP	String showing the build host name followed by a time stamp.
Install Date	N/A	Date and time values that indicate when the operating system was installed.
Package Status	N/A	One of three values that indicate whether or not a package is fully installed: `unknown`, `completely installed`, `partially installed`.
Support Information	HOTLINE	Character string that provides information about who to call for support.

Patches

The `Solaris_Application1.0.mof` file also enables you to provide fixes to problems and updated versions of your applications in the form of patches. The following table lists and describes the patch attributes that you can provide.

TABLE B–3 Patch Information You Can Provide

Patch Attribute	Description
Obsolete	Provides a description of patches that are obsolete or that have been integrated into the current patch.
Required	Lists the patches required to make the current patch work.
Incompatible	Lists patches that conflict with the current patch.
Manufacturer	Lists the manufacturer's name.
Installation Date	Provides a date/time value indicating the date and time that the patch was installed.

The `Solaris_System1.0.mof` File

The `Solaris_System1.0.mof` file defines the `Solaris_Process` and `Solaris_OperatingSystem` classes.

```
// ==================================================================
// Solaris_Process
// ==================================================================

 [Provider("Solaris"),
  Description ("A Solaris process that is running.")
 ]
class Solaris_Process:CIM_Process
{
};

// ==================================================================
// Solaris_OperatingSystem
// ==================================================================

 [Provider("Solaris"),
  Description ("The Solaris Operating System.")
 ]
```

(continued)

(Continuation)

```
class Solaris_OperatingSystem:CIM_OperatingSystem
{
};
```

The Solaris_Device1.0.mof File

The Solaris_Device1.0.mof file defines the following classes:

- Solaris_Processor class as an extension of CIM_Processor.
- Solaris_DiskDrive class as an extension of CIM_DiskDrive.
- Solaris_SerialPort is defined as an extension of CIM_SerialController. Its configuration class, Solaris_SerialPortConfiguration is defined as an extension of CIM_Setting.

Solaris_Processor

Solaris_Processor is defined as an extension of CIM_Processor with the following properties.

```
class Solaris_Processor:CIM_Processor
{
    string Name;
    string Description;
    string Architecture;
    string ClockSpeed;
    string SparcVersion;
    uint32 D_Cache;
    uint32 E_Cache;
    uint32 I_Cache;
};
```

Solaris_DiskDrive

Currently, the class definition is provided for `Solaris_DiskDrive`.

```
]
class Solaris_DiskDrive:CIM_DiskDrive
{
```

Solaris_SerialPort

The `Solaris_SerialPort` class is defined with Boolean properties that enable you to control how serial port characteristics, such as baud rate and parity, are handled by the port.

```
// ==================================================================
// Solaris_SerialPort
// ==================================================================

 [Provider("com.sun.wbem.solarisprovider.Solaris"),
  Description (
              "This is the MOF file that defines a Solaris serial port.")
 ]
class Solaris_SerialPort:CIM_SerialController
{
 [read]
    boolean ServiceEnabled;
 [read]
    boolean SettableBaudRate;
        [read]
    boolean SettableDataBits;
        [read]
    boolean SettableFlowControl;
        [read]
    boolean SettableParity;
        [read]
    boolean SettableParityCheck;
        [read]
    boolean SettableStopBits;
        [read]
    boolean SupportsParityCheck;
        [read]
    boolean SupportsXOnXOff;
        [read]
    boolean SupportsXOnXOffSet;
 [read]
```

(continued)

(Continuation)

```
    string PortMonitor;
 [read]
    string ServiceTag;
 [read]
    string Comment;
    boolean DisablePortService();
};
```

Solaris_PortConfiguration

`Solaris_PortConfiguration` properties enable you to specify data values that a user can view or modify.

```
[Provider ("Solaris") ]
class Solaris_SerialPortConfiguration:CIM_Setting
{
[read, write]
boolean ServiceEnabled;
[read, write]
uint32 BaudRate;
[read, write]
string TerminalType;
[read, write]
boolean TTYFlag_Init;
[read, write]
boolean TTYFlag_Bidirectional;
[read, write]
boolean TTYFlag_CarrierConnect;
[read, write]
boolean SoftwareCarrier;
[read, write]
boolean CreateUtmp;
[read, write]
string LoginPrompt;
[read, write]
string Comment;
[read, key]
string ServiceTag;
[read, key]
string PortName;
[read]
string deviceName;
[read, write]
string PortmonTag;
[read, write]
string ServiceProgram;
[read, write]
```

(continued)

(Continuation)

```
string StreamsModules;
[read, write]
string Timeout; };
```

The `Solaris_Acl1.0.mof` File

This file specifies the Solaris WBEM Services security classes. It defines the base classes for access control lists, users, and namespaces in the Solaris operating environment. For information about this file, see “Using the APIs to Set Access Control” on page 38 in Chapter 3. For information about Solaris WBEM Services security features, see Chapter 3.

Glossary

This Glossary defines terms used in the Solaris WBEM Services documentation. Many of these terms are familiar to developers, but have new or altered meaning in the WBEM environment.

alias A symbolic reference in either a class or instance declaration to an object located elsewhere in a MOF file. Alias names follow the same rules as instance and class names. Aliases are typically used as shortcuts to lengthy paths.

aggregation relationship A relationship in which one entity is made up of the aggregation of some number of other entities.

association class A class that describes a relationship between two classes or between instances of two classes. The properties of an association class include pointers, or references, to the two classes or instances. All WBEM classes can be included in one or more associations.

Backus-Naur Form (BNF) A metalanguage that specifies the syntax of programming languages.

cardinality The number of values that may apply to an attribute for a given entity.

class A collection or set of objects that have similar properties and fulfill similar purposes.

CIM Object Manager Repository A central storage area managed by the Common Information Model Object Manager (CIM Object Manager). This repository contains the definitions of classes and instances that represent managed objects and the relationships among them.

CIM Schema A collection of class definitions used to represent managed objects that occur in every management environment.

See also core model, common model, and extension schema.

The CIM is divided into the metamodel and the standard schema. The metamodel describes what types of entities make up the schema. It also defines how these entities can be combined into objects that represent managed objects.

common model The second layer of the CIM schema, which includes a series of domain-specific but platform-independent classes. The domains are systems, networks, applications, and other management-related data. The common model is derived from the core model.

See also extension schema.

core model The first layer of the CIM schema, which includes the top-level classes and their properties and associations. The core model is both domain- and platform-independent.

See also common model and extension schema.

Distributed Management Task Force (DMTF) An industry-wide consortium committed to making personal computers easier to use, understand, configure, and manage.

domain The class to which a property or method belongs. For example, if status is a property of Logical Device, it is said to belong to the Logical Device domain.

dynamic class A class whose definition is supplied by a provider at runtime as needed. Dynamic classes are used to represent provider-specific managed objects and are not stored permanently in the CIM Object Manager Repository. Instead, the provider responsible for a dynamic class stores information about its location. When an application requests a dynamic class, the CIM Object Manager locates the provider and forwards the request. Dynamic classes support only dynamic instances.

dynamic instances An instance that is supplied by a provider when the need arises and is not stored in the CIM Object Manager Repository. Dynamic instances can be provided for either static or dynamic classes. Supporting instances of a class dynamically allows a provider to always supply up-to-the-minute property values.

enumeration Java term for getting a list of objects. Java provides an `Enumeration` interface that has methods for enumerating a list of objects. An individual object on this list to be enumerated is called an element.

extension schema The third layer of the CIM Schema, which includes platform-specific extensions of the CIM Schema such as Solaris and UNIX.

See also common model and core model.

flavor *See* qualifier flavor.

indication An operation executed as a result of some action such as the creation, modification, or deletion of an instance, access to an instance, or modification or access to a property. Indications can also result from the passage of a specified period of time. An indication typically results in an event.

inheritance The relationship that describes how classes and instances are derived from parent classes or superclasses. A class can spawn a new subclass, also called a child class. A subclass contains all the methods and properties of its parent class. Inheritance is one of the features that allows WBEM classes to function as templates for actual managed objects in the WBEM environment.

instance A representation of a managed object that belongs to a particular class, or a particular occurrence of an event. Instances contain actual data.

instance provider A type of provider that supports instances of system- and property-specific classes. Instance providers can support data retrieval, modification, deletion, and enumeration. Instance providers can also invoke methods.

See also property provider.

interface class The class used to access a set of objects. The interface class can be an abstract class representing the scope of an enumeration.

See also enumeration and scope.

Interface Definition Language (IDL) A generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

key A property that is used to provide a unique identifier for an instance of a class. Key properties are marked with the Key qualifier.

Key qualifier A qualifier that must be attached to every property in a class that serves as part of the key for that class.

managed object — A hardware or software component that is represented as a WBEM class. Information about managed objects is supplied by data and event providers as well as the CIM Object Manager Repository.

Managed Object Format (MOF) — A compiled language for defining classes and instances. The MOF compiler (`mofc`) compiles `.mof` text files into Java classes and adds the data to the CIM Object Manager Repository. MOF eliminates the need to write code, thus providing a simple and fast technique for modifying the CIM Object Manager Repository.

management application — An application or service that uses information originating from one or more managed objects in a managed environment. Management applications retrieve this information through calls to the CIM Object Manager API from the CIM Object Manager and from providers.

management information base — A database of managed objects.

metamodel — A CIM component that describes the entities and relationships representing managed objects. For example, classes, instances, and associations are included in the metamodel.

metaschema — A formal definition of the Common Information Model, which defines the terms used to express the model, its usage, and its semantics.

method — A function describing the behavior of a class. Including a method in a class does not guarantee an implementation of the method.

MOF file — A text file that contains definitions of classes and instances using the Managed Object Format (MOF) language.

Named Element — An entity that can be expressed as an object in the metaschema.

namespace — A directory-like structure that can contain classes, instances, and other namespaces.

object path — A formatted string used to access namespaces, classes, and instances. Each object on the system has a unique path which identifies it locally or over the network. Object paths are conceptually similar to Universal Resource Locators (URLs).

override — Indicates that the property, method, or reference in the derived class overrides the similar construct in the parent class in the inheritance tree or in the specified parent class.

polymorphism The ability to alter methods and properties in a derived class without changing their names or altering interfaces. For example, a subclass can redefine the implementation of a method or property inherited from its superclass. The property or method is thereby redefined even if the superclass is used as the interface class.

Thus, the LogicalDevice class can define the variable status as a string, and can return the values "on" or "off." The Modem subclass of LogicalDevice can redefine (override) status by returning "on," "off," and "connected." If all LogicalDevices are enumerated, any LogicalDevice that happens to be a modem can return the value "connected" for the status property.

property A value used to characterize the instances of a class. Property names cannot begin with a digit and cannot contain white space. Property values must have a valid Managed Object Format (MOF) data type.

property provider A program that communicates with managed objects to access data and event notifications from a variety of sources, such as the Solaris operating environment or a Simple Network Management Protocol (SNMP) SNMP device. Providers forward this information to the CIM Object Manager for integration and interpretation.

qualifier A modifier containing information that describes a class, an instance, a property, a method, or a parameter. The three categories of qualifiers are: those defined by the Common Information Model (CIM), those defined by WBEM (standard qualifiers), and those defined by developers. Standard qualifiers are attached automatically by the CIM Object Manager.

qualifier flavor An attribute of a CIM qualifier that governs the use of a qualifier. WBEM flavors describe rules that specify whether a qualifier can be propagated to derived classes and instances and whether or not a derived class or instance can override the qualifier's original value.

range A class that is referenced by a reference property.

reference A special string property type that is marked with the reference qualifier, indicating that it is a pointer to other instances.

required property A property that must have a value.

schema A collection of class definitions that describe managed objects in a particular environment.

scope	An attribute of a CIM qualifier that indicates which CIM elements can use the qualifier. Scope can only be defined in the Qualifier Type declaration; it cannot be changed in a qualifier.
selective inheritance	The ability of a descendant class to drop or override the properties of an ancestral class.
Simple Network Management Protocol (SNMP)	A protocol of the Internet reference model used for network management.
singleton class	A WBEM class that supports only a single instance.
Solaris Schema	A Sun extension to the CIM Schema that contains definitions of classes and instances to represent managed objects that exist in a typical Solaris operating environment.
standard schema	A common conceptual framework for organizing and relating the various classes representing the current operational state of a system, network, or application. The standard schema is defined by the Distributed Management Task Force (DMTF) in the Common Information Model (CIM).
static class	A WBEM class whose definition is persistent. The definition is stored in the CIM Object Manager Repository until it is explicitly deleted. The CIM Object Manager can provide definitions of static classes without the help of a provider. Static classes can support either static or dynamic instances.
static instance	An instance that is persistently stored in the CIM Object Manager Repository.
subclass	A class that is derived from a superclass. The subclass inherits all features of its superclass, but can add new features or redefine existing ones.
subschema	A part of a schema owned by a particular organization. The Win32 and Solaris Schemas are examples of subschemas.
superclass	The class from which a subclass inherits.
transitive dependency	In a relation having at least three attributes R (A, B, C), the situation in which A determines B, B determines C, but B does not determine A.

trigger	A recognition of a state change (such as create, delete, update, or access) of a class instance, and update or access of a property. The WBEM implementation does not have an explicit object representing a trigger. Triggers are implied either by the operations on basic objects of the system (create, delete, and modify on classes, instances and namespaces) or by events in the managed environment.
Unified Modeling Language (UML)	A notation language used to express a software system using boxes and lines to represent objects and relationships.
Unicode	A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.
UTF-8	An 8-bit transformation format that may also serve as a transformation format for Unicode character data.
virtual function table (VTBL)	A table of function pointers, such as an implementation of a class. The pointers in the VTBL point to the members of the interfaces that an object supports.
Win32 Schema	A Microsoft extension to the CIM Schema that contains definitions of classes and instances to represent managed objects that exist in a typical Win32 environment.

Index