WDR™ Developer's Guide

Creating System Management Applications

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

Part No. 816-1984-10
September 2002

Send comments about this document to: docfeedback@sun.com

Please Recycle

Adobe PostScript

Contents

Preface

This *WDR Developer's Guide* is intended for use by systems administrators who want to develop applications that perform DR operations remotely using WBEM, which is an industry standard for Web-based enterprise management.

Developers can write WDR client applications in languages such as Java, using software development kits (SDKs) such as the Sun WBEM SDK.

Before You Read This Book

This book is intended for the Sun Fire 15K, 12K, 6800, 4810, 4800, and 3800 system platform administrator who has a working knowledge of UNIX® systems, particularly those based on the Solaris™ operating environment. If you do not have such knowledge, first read the Solaris user and system administrator books provided with this system, and consider UNIX system administration training.

How This Book Is Organized

Chapter 1 “Introduction to DR” provides an overview of WDR, and describes the kind of tasks that WDR enables you to perform.

Chapter 2 “Using Solaris WBEM Services in WDR” describes the different layers in Solaris WBEM Services, which are included in the Solaris operating environment.

Chapter 3 “Using Process Indications” describes process indications, which are notifications of system events to which each WDR client can subscribe.

Chapter 4 “Classes, Domains, Associations, Indications in WDR” introduces all the classes, indications (of system events), and associations that WDR provides to the developer. All methods and properties that the developer needs to use are described in this chapter.

Chapter 5 “Programming Techniques in WDR” presents programming techniques that the developer may find useful in creating WDR applications that simplify and automate systems administration on Sun Fire 3800, 4800, 4810, 6800, 15K and 12K systems.

Using UNIX Commands

This document does not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- *Solaris Handbook for Sun Peripherals*
- Online documentation for the Solaris™ operating environment
- Other software documentation that you received with your system

Typographic Conventions

TABLE P-1

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

Shell Prompts

TABLE P-2

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

Related Documentation

TABLE P-3

Application	Title	Part Number
WDR Installation	*WDR Installation Guide*	816-4820
DR on Sun Fire 6800, 4810, 4800, and 3800 systems	*Sun Fire 6800, 4810, 4800, and 3800 Systems Dynamic Reconfiguration User Guide*	806-6783
DR on Sun Fire 15K and 12K systems	*Sun Fire 15K/12K Dynamic Reconfiguration User Guide*	816-5075
System-level security on Sun Fire 15K and 12K systems	*System Management Services (SMS) 1.2 Administrator Guide for Sun Fire 15K/12K Systems*	816-5259
System-level security on Sun Fire 6800 / 4810 / 4800 / 3800 systems	*Sun Fire 6800/4810/4800/3800 Systems Platform Administration Manual*	805-7373
Solaris WBEM Services	*Solaris WBEM Services Administrator's Guide*	806-6468

Accessing Sun Documentation Online

The `docs.sun.com`[sm] web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

```
http://docs.sun.com
```

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

```
docfeedback@sun.com
```

Please include the part number (816-1984-10) of this document in the subject line of your email.

CHAPTER 1

Introduction to WDR

WDR (WBEM dynamic reconfiguration) provides an application program interface (API) that software applications can use to perform dynamic reconfiguration (DR) operations remotely on the following systems:

- Sun Fire 3800
- Sun Fire 4800
- Sun Fire 4810
- Sun Fire 6800
- Sun Fire 15K
- Sun Fire 12K

Software developers and systems administrators can use the WDR API to create custom applications that remotely perform crucial system management functions such as load balancing. WDR provides an alternative to the current, conventional method of performing DR operations, which are achieved either on the Sun Fire System Controller (SC) or on the Solaris domain (using the `cfgadm` system library).

Hardware Required for WDR

On Sun Fire 6800/4810/4800/3800 systems, WDR runs on an external host that is referred to as the *Midframe Service Processor* (MSP). On Sun Fire 15K and 12K systems, WDR runs on the System Controller (SC).

Hardware Required for MSP on Sun Fire 6800/4810/4800/3800 Systems

The minimum hardware requirements for an MSP are:

- sun4u architecture

- 8 GB disk space
- 128 MB RAM
- CD-ROM drive
- SunSwift™ card or, ideally, a QuadFast Ethernet card

Software Required for WDR

WDR can be used on Sun Fire 3800/4800/4810/6800 and Sun Fire 15K/12K system domains that run the Solaris 8 2/02 and Solaris 9 software. WDR is not bundled with other software, such as the Solaris operating environment

Software Required for Sun Fire 15K and 12K Systems

To enable WDR, both the WDR software and Solaris WBEM Services software must be installed on the SC. Further, the System Management Services (SMS) version 1.2 software must be installed on the SC.

Software Required for Sun Fire 3800, 4800, 4810, and 6800 Systems

To enable WDR, both the WDR software and Solaris WBEM Services software must be installed on the MW.

About Web-Based Enterprise Management (WBEM)

The WDR interface is based on the Web-based Enterprise Management (WBEM) industry standard, which enables Web-based management of systems, networks, and devices on a variety of platforms. WBEM was developed by members of the Distributed Management Task Force (DMTF), who represent many industry leaders.

WBEM is comprised of three principal components:

- A method of modeling managed objects. WBEM uses the Common Information Model (CIM) to create classes that represent managed objects. These classes have properties that represent the attributes and states of managed objects; and methods that represent operations that can be performed on managed objects.
- A means of encoding CIM information so that it can be sent over the wire. WBEM uses Extensible Markup Language (XML), a powerful and extensible subset of SGML, to encode CIM classes.
- A way of encapsulating XML operations for transmission over the wire. WBEM uses XML / HTTP or RMI for sending operations that get information from, set the properties of, and perform operations on, managed objects

To summarize: in WBEM, managed objects are represented as CIM classes, properties, and methods; CIM operations are represented as either XML / HTTP or RMI messages; and those messages are sent over the wire.

A comprehensive description of the WBEM standard is beyond the scope of this document. However, complete information about WBEM is available from a variety of sources, including the DMTF Web site at `www.dmtf.org`.

Common Information Model (CIM)

WDR is a Sun Fire system-specific extension of the CIM schema that is used to represent:

- Resources on Sun Fire systems that can be managed using DR,
- Events that relate to DR or affect the state of the WDR model,
- DR platform resources such as attachment points, which are represented by the AttachmentPoint class and its subclasses,
- The containers of DR platform resources, such as domains and slots,
- Events that affect the existence and / or state of objects in the WDR schema,
- Associations between objects in the WDR schema, and
- DR operations themselves.

The architecture of the Sun Fire 3800 / 4800 / 4810 / 6800 systems differs significantly from that of the Sun Fire 15K and 12K systems. WDR includes CIM schema that reflect the architectures of all the different Sun Fire systems on which it is used.

Some of the objects in the CIM schema are common to all Sun Fire systems; other objects are used only on the Sun Fire 3800 / 4800 / 4810 / 6800 systems; while other objects are used only on the Sun Fire 15K and 12K systems.

The *commonalities* between the system architectures are captured in platform-independent superclasses; the *differences* are captured in platform-specific subclasses of those platform-independent superclasses.

Platform-Specific and Common MOF Files

The CIM schema used by WDR is expressed in three Managed Object Format (MOF) files, which are ASCII text files that define all the objects that represent managed resources on Sun Fire systems.

One MOF file, `WDR_core1.0.mof`, defines the common elements of all Sun Fire systems. The other two MOF files define system-specific elements:

- `WDR_XC1.0.mof` defines elements specific to Sun Fire 15K and 12K systems.
- `WDR_SG1.0.mof` defines elements specific to the Sun Fire 6800/4810/4800/3800 systems.

In addition to providing a schema, the MOF file also provides the software developer or systems administrator with a formal definition of the objects that comprise the WDR CIM schema.

Note – For a formal definition of CIM, see *Common Information Model, Implementing the Object Model for Enterprise Management*, Winston Bumpus et al., Wiley Computer Publishing, copyright 2000, New York, ISBN 0-471-35342-6.

Operations that WDR Performs

WDR can perform the following dynamic reconfiguration operations remotely:

- Add a system board (a CPU/memory board) to a domain that is running the Solaris software. DR first connects the board electrically to the system, putting it into a *connected* state. DR then configures the system board so that it is fully available to all applications running in the domain; the board is put into the *configured* state.
- Move a system board from one domain to another domain, via an unconfigure operation followed by a configure operation.
- Remove a system board from a domain and make it available for use by other domains.
- List all attachment points that are currently available to domains on the system.
- Display information about the current state of a specified system board, such as its power status, availability, and domain assignment.
- Retrieve the memory configuration of a configured system board.
- Retrieve information about the impact on memory, such as memory drain information, that is associated with detaching a configured system board.

The functionality of WDR is the same as the underlying functionality of DR itself; WDR adds no additional operations to DR. However, WDR does enhance DR by providing information about domains and slots; associations between classes; and event notification.

WDR is designed to perform DR operations efficiently, without any noticeable degradation of performance.

Administrator Security Models

WDR enforces the administrator security models on Sun Fire 15K, 12K, 6800, 4810, 4800, and 3800 systems.

For complete information about implementing security at the Sun Fire 6800/4810/4800/3800 system level, see the *Sun Fire 6800/4810/4800/3800 Systems Platform Administration Manual* (part number 805-7373).

For complete information about implementing security at the Sun Fire 15K/12K system level, see the *System Management Services (SMS) 1.2 Administrator Guide for Sun Fire 15K/12K Systems* (part number 816-5259).

In addition, security that is available through Solaris WBEM Services is described in Chapter 2 “Using Solaris WBEM Services in WDR.”

WDR Security

The `/etc/group` file shows the groups to which the currently logged in user is subscribed.

Sun Fire 6800, 4810, 4800, and 3800 System Groups

The `/etc/group` file, which shows group membership on a Sun Fire 6800/4810/4800/3800 system, can be edited manually.

The following table shows all the operations that users can perform based on their group membership:

TABLE 1-1 Permitted Tasks Based on Group - Sun Fire 6800/4810/4800/3800

Group	Tasks that the User Can Perform
None (all users)	Enumerate domains and slots
`spltadm`	Assign and unassign boards
`spltop`	No special privileges
`sd`*x*`adm`	Where *x* represent a domain, can: • Enumerate attachment points in domain *x*. • Enumerate all attachment points if the user is in the `sd`*x*`adm` group in all domains. • Change an attachment point state, assign, unassign, power-on, and power-off a board that is in domain *x*'s ACL.
`sd`*x*`op`	Where *x* represent a domain, can: • Enumerate attachment points in domain *x*. • Enumerate all attachment points if the user is in the `sd`*x*`op` group in all domains.

Sun Fire 15K and 12K System Groups

To modify the `/etc/group` file, which shows group membership on a Sun Fire15K or 12K system, you run the /opt/SUNWSMS/bin/smsconfig script with arguments. See the *System Management Services (SMS) 1.2 Administrator Guide for Sun Fire 15K/12K Systems* for more information.

The following table shows all the operations that users can perform based on their group membership:

TABLE 1-2 Permitted Tasks Based on Group - Sun Fire 15K and 12K

Group	Tasks that the User Can Perform
`platadmn`	Assign, unassign, power-on, and power-off boards

TABLE 1-2 Permitted Tasks Based on Group - Sun Fire 15K and 12K

Group	Tasks that the User Can Perform
`platoper`	No special privileges
`dmn`*x*`adm`	Where *x* represent a domain, can: • Enumerate attachment points in domain *x*. • Enumerate all attachment points if the user is in the `dmn`*x*`adm` group in all domains. • Change an attachment point state, assign, unassign, power-on, and power-off a board that is in domain *x*'s ACL.
`dmn`*x*`rcfg`	Where *x* represent a domain, can: • Enumerate attachment points in domain *x*. • Enumerate all attachment points if the user is in the `dmn`*x*`rcfg` group in all domains. • Change an attachment point state, assign, unassign, power-on, and power-off a board that is in domain *x*'s ACL.

Solaris WBEM Services

WDR is an extension of the Solaris WBEM Services software, which is included in both the Solaris 8 2/02 and Solaris 9 operating environments. Solaris WBEM Services software provides secure access and manipulation of management data, and enables software developers to create client applications that manage system resources in the Solaris environment.

Solaris WBEM Services software consists of components that function at three levels:

- The Application Layer, where WBEM clients process and display data from managed resources. Application Layer services includes the WBEM Workshop; the WBEM User Manager, which allows administrators to add and remove authorized WBEM users and set their access privileges; and the MOF compiler.
- The Management Layer, where the CIM API (which forms the boundary between the Application and Management Layers) enables the administrator to perform operations such as viewing and creating classes and instances of managed resources from the CIMOM. The CIMOM, the CIM Repository, and the Provider interface all reside at the Management Layer.
- The Provider Layer. At this layer resides the Solaris Provider, which provides the CIMOM instances of managed resources in the Solaris operating environment, and gets and sets information about managed resources. The Solaris Provider forms the interface between CIMOM and managed system resources.

Solaris WBEM Services components interact with the Solaris software and with the system hardware. For more information about the Solaris WBEM Services software, visit the Solaris WBEM Web site at `www.sun.com/software/solaris/wbem`.

Developers of load balancing and other system management applications can use Solaris WBEM Services software to obtain information about the current level of resource utilization on a Sun Fire system domain. WDR itself does not provide system performance data.

CIM Object Manager (CIMOM)

The CIMOM manages CIM objects on a WBEM system. The CIMOM transfers information between WBEM clients, the CIMOM Repository, and to managed resources via providers. The CIMOM accepts connections from management applications using the RMI protocol, and provides the following services to connected clients:

- Management services. The CIMOM 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 that developers can use to create applications that dynamically record CIMOM event data to, and retrieve it from, a log record.
- XML services that convert XML data into CIM classes, which enables XML-based WBEM clients to communicate with the CIMOM.

WBEM Providers

WDR contains several provider classes, which are expressed in the MOF files. WBEM providers are classes that act as intermediaries between the CIMOM and managed objects on a system. WBEM providers get information from, set information on, and may perform operations on, managed devices. WBEM providers forward retrieved information to the CIMOM, which is a part of the Solaris WBEM Services software, for delivery to the requesting clients.

When the CIMOM receives a request for information that is not available in the CIMOM Repository, it forwards the request to a provider. The provider receives requests for information, and returns the information, using APIs.

Solaris WBEM Software Development Kit (SDK)

Developers of WDR applications can use the Solaris WBEM SDK. However, there is no requirement to use the Solaris WBEM SDK because WDR uses a standard set of protocols. For more information about the Solaris WBEM SDK visit the Sun Developer Connection at:

`www.sun.com/solaris/wbem`

CHAPTER 2

Using Solaris WBEM Services in WDR

Overview of Solaris WBEM Services

Solaris WBEM Services provide the WDR application developer with a variety of WBEM services on domains that are running either the Solaris 8 2/02 or Solaris 9 operating environment. Solaris WBEM Services, which are included with the Solaris software, make it easier for developers to create applications that use WBEM to manage systems running Solaris software.

This developer's guide provides information about only those Solaris WBEM Services with which a WDR application developer needs to become familiar. Complete information about Solaris WBEM Services is available at the following Web site:

`http://www.sun.com/solaris/wbem`

Solaris WBEM Services provide secure access to information about managed resources, which in turn enable applications that use WDR to get information about, and manage, system resources. A built-in Solaris Provider allows access to information about managed resources such as hardware and software state information, performance metrics, and other data that are needed by management applications to perform load balancing and to respond to device failovers.

Solaris WBEM Services uses the Common Information Model (CIM) to create a schema that represents managed objects in a system running Solaris software. CIM objects are specified in a Managed Object Format (MOF) file, which is provided with WDR and compiled when WDR is installed.

Layers of Solaris WBEM Services

Solaris WBEM Services is a software package that resides at three layers. At each layer reside software components that are important to WDR application developers:

- Application Layer
- Management Layer
- Provider Layer

Solaris WBEM Services Application Layer

The following Solaris WBEM Services Application Layer software programs, which are especially useful to WDR application developers, are described in detail in this chapter:

- Solaris Management Console (SMC) WBEM Log Viewer on page 13
- Managed Object Format (MOF) Compiler on page 13
- Sun WBEM User Manager on page 21
- Solaris Management Console (SMC) Users Tool on page 27

Sun WBEM User Manager and SMC Users Tool

The Sun WBEM User Manager and SMC Users Tool applications enable systems administrators to add and remove authorized users and to set their access privileges to managed resources.

There are two separate mechanisms for administering security with domains running the Solaris software: WBEM access control list (ACL) and Solaris role-based access control (RBAC).

You use the WBEM User Manager to add users to existing access control lists (ACLs) and to grant them either read or read-write access privileges.

You use the Users Tool in the Solaris Management Console (SMC) to add users, and to grant user roles and privileges, using RBAC.

See the section “WBEM Security Services” on page 19 for more information about administering WBEM security, including details of ACL- and RBAC-based system security.

Solaris Management Console (SMC) WBEM Log Viewer

The SMC WBEM Log Viewer displays log files that include information such as the names of users who issued logged commands, and the client computers on which the logged commands were issued.

Solaris WBEM Services includes APIs to enable logging of system events. See the section “Solaris WBEM Logging Services” on page 28 (and subsequent sections) for complete information about log files; rules associated with log files; log file formats;, classes that developers can use to record system events; and using APIs to enable and use logging services.

Managed Object Format (MOF) Compiler

The MOF Compiler is used to compile MOF files, which are ASCII text files that specify objects in a CIM schema that represent managed objects in a system running Solaris software.

WDR includes three MOF files that define schema comprised of objects that represent managed resources. One MOF file is used for all Sun Fire systems; another is used only on Sun Fire 15K and 12K systems; and the third is used for Sun Fire 3800, 4800, 4810, or 6800 systems.

The MOF compiler reads statements in a MOF file that define classes and instances, and then adds them to the CIM Object Manager Repository, which is a central storage area for information about management data.

The `mofcomp` Command

To start the MOF compiler and compile a MOF file, use the `mofcomp` command:

```
/usr/sadm/bin/mofcomp [-help] [-v] [-sc] [-si] [-sq] [-version]
[-c cimom_hostname] [-u username] [-p password] filename
```

Where:

TABLE 2-1 Arguments to the `mofcomp` Command

Argument	Description
`-help`	Lists the arguments to the `mofcomp` command.
`-v`	Runs the compiler in verbose mode, which displays all compiler messages.
`-sc`	Runs the compiler with the "set class" option, which updates a class if it already exists and contains no instances, and returns an error if the class does not already exist. If you do not specify the `-sc` option, the compiler adds a CIM class to the connected namespace, and returns an error if the class already exists.
`-si`	Runs the compiler with the "set instance" option, which updates an instance if it already exists, and returns an error message if it does not. If you do not specify the `-si` option, the compiler adds a CIM instance to the connected namespace, and returns an error if the instance already exists.
`-sq`	Runs the compiler with the "set qualifier types" option, which updates a qualifier if it already exists, and returns an error message if it does not. If you do not specify the `-sq` option, the compiler adds a CIM qualifier type to the connected namespace, and returns an error if the qualifier type already exists.
`-`*`version`*	Displays the version number of the MOF compiler.
`-c` *`cimom_hostname`*	Specifies a system that is running the CIM Object Manager.

TABLE 2-1 Arguments to the `mofcomp` Command

Argument	Description
`-u` *`username`*	Specifies the user name for connecting to the CIM Object Manager. Use the `-u` *`username`* 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 you for the password. See the section "mofcomp Password Security Advisory" on page 16 below.
`-p` *`password`*	Specifies 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 you for the password. See the section "mofcomp Password Security Advisory" on page 16 below.
`filename`	The name of the MOF file to be compiled.

Compiling a MOF File

You can compile a MOF file whether its filename contains or does not contain 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 with no options, type the following:**

   ```
   # mofcomp filename
   ```

   For example,

   ```
   # mofcomp /usr/sadm/mof/Solaris_Application1.0.mof
   ```

   The MOF file named `Solaris_Application1.0.mof` is compiled into the CIM Object Manager Repository.

`mofcomp` Password Security Advisory

If you run the `mofcomp` command with the `-p` option, or with the `-p` and `-u` options, and you include a password on the command line, another user can subsequently run the `ps` command or the `history` command to display your password. The system does not display a security warning.

Note – If you run a command that requires you to provide your password on the command line, immediately change your password after running the command. This will prevent another user from displaying your current password.

The following examples show unsafe (insecure) usage:

```
% mofcomp -p Log8Rif
```

```
% mofcomp -up molly Log8Rif
```

If you use the `mofcomp` command in either of the preceding ways, make sure to change your password immediately after running the command.

Solaris WBEM Services Management Layer

The Solaris WBEM Services Management Layer software program that is useful to WDR application developers is the Common Information Model (CIM) Object Manager.

CIM Object Manager

Solaris WBEM Services includes the CIM Object Manager, which manages objects in a WBEM-enabled system. Each CIM object represents a managed system object, such as a CPU, an I/O board, or an attachment point.

The CIM Object Manager first accepts connections to management applications using either the RMI or XML/HTTP protocol; sets up a connection to the CIM Object Repository; and then awaits requests from client applications for services, which include:

- Management services that check the semantics and syntax of CIM data operations to ensure that they comply with the latest CIM specification; and that distribute management data between applications (such as WDR applications), the CIM Repository, and managed resources.
- Security services that authenticate user login requests and control access to system resources.
- Logging services that record system events

After WBEM clients are connected to a WBEM-enabled system, they can request WBEM operations such as creating, viewing, and deleting CIM classes and instances; retrieving the values of properties; and enumerating instances of classes, or classes within a specified class hierarchy.

Manually Starting and Stopping the CIM Object Manager

Normally, the CIM Object Manager is started automatically during installation and whenever you boot a domain by a utility called `/etc/init.d/init.wbem`. In addition to the CIM Object Manager, the command starts the Solaris Management Console (SMC); both run as a single process.

You should not need to start and stop the CIM Object Manager manually, but you can do so if the need should arise. The `init.wbem` utility has the following syntax:

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

The `start` option starts the CIM Object Manager on the domain from which it is invoked. The `stop` option stops the CIM Object Manager on the domain. The `status` option gets the status of the CIM Object Manager on the domain.

▼ To Start the CIM Object Manager

1. **Enter the following command at the system prompt to become a root user:**

   ```
   % su
   ```

2. **At the root system prompt (#) type the root password for the domain when prompted to do so.**

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

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

▼ To Stop the CIM Object Manager

1. **Enter the following command at the system prompt to become a root user:**

   ```
   % su
   ```

2. **When prompted, enter the root password for the domain at the root system prompt (#).**

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

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

Solaris WBEM Services Provider Layer

The Solaris WBEM Services Provider Layer includes the Solaris Provider software program, which is especially useful to WDR application developers.

Solaris Providers

A Solaris Provider is a class that communicates with managed objects. Providers provide the CIM Object Manager with instances of managed resources on systems running the Solaris operating environment, and retrieve and set information on managed devices.

When a WDR application attempts to access CIM data about managed resources, WBEM first validates the user login information on the domain. Users are granted Read Only access by default. See the section “WBEM Security Services” on page 19 for more information about WBEM system security.

The CIM Object Manager uses object provider APIs to communicate with providers. After an application requests dynamic data from the CIM Object Manager, the CIM Object Manager responds via the provider APIs to pass the requested information to the provider.

Providers can be either native providers, which are machine-specific, or they can be written using the portable, machine-independent Java Native Interface (JNI), which is part of the Java Development Kit (JDK).

WBEM Security Services

There are three principal security features that protect CIM objects from intrusion on a WBEM-enabled system:

- Authentication
- Authorization
- Replay protection

Authentication

Authentication is the process of verifying the identity of a user, device, or other entity in a Sun Fire system. Authentication is frequently used to give valid users access to system resources; and to deny access to users who cannot be authenticated.

When a user logs in and enters a user name and password, the client uses the password to generate an encrypted digest that the server verifies. When the user is authenticated, the CIM Object Manager grants a MAC token and establishes a client session. All subsequent operations occur within that secure client session, and contain a MAC token that uses the session key that was negotiated during the authentication process. (A MAC is a token parameter added to a remote call which contains security information used to authenticate that message.)

Authorization

Authorization is the process of granting to a user, program, or process the right to access system resources. Authorization occurs after authentication.

After 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 an application or any of its related tasks. The CIM Object Manager supports capability-based authorization, which allows a privileged user to assign read and write access to other users. Such authorizations are added to existing Solaris user accounts.

Replay Protection

Replay protection prevents an unauthorized client picking up and sending another client's message to the server by validating a session key.

A client cannot copy another client's last message that was sent to the CIM Object Manager. The CIM Object Manager uses a MAC for each message, based on the session key that was negotiated during authentication, to guarantee that all communications in the client-server session is indeed with the same client that initiated the session and participated in client-server authentication.

The MAC is used to confirm that each message actually came from the client that was originally authenticated for the session, and that the message was not being replayed by another client. This type of mechanism is used in WBEM to verify RMI messages. The session key that was negotiated during the user authentication exchange is used to encrypt the security information in the message's MAC token.

Digital Signatures

WBEM Security Services does not perform digital signing of messages.

Implementing Security

You use WBEM access control lists (ACLs) to administer security within the Solaris operating environment.

WBEM Access Control Lists

ACL-based security is implemented using classes that are defined in the `Solaris_Acl1.0.mof` file. ACL-based security, which is specific to Solaris WBEM Services, provides a default authorization scheme for Solaris WBEM Services, and applies to all CIM operations. Instances of these classes determine the default authorizations that are assigned to WBEM users and/or namespaces.

To add users to existing ACLs and assign to them either read or read-write access privileges, use the Sun WBEM User Manager, which is described in the section Sun WBEM User Manager. The Sun WBEM User Manager is located at `/usr/sadm/bin/wbemadmin`.

For more information, see the section "Sun WBEM User Manager" on page 21.

Sun WBEM User Manager

The Sun WBEM User Manager allows privileged users to add and delete authorized users and to set their access privileges to CIM objects on a WBEM-enabled system. Each user must have a Solaris user account.

You can use the Sun WBEM User Manager to set access privileges on individual namespaces or on a user/namespace combination. When you add a user and select a namespace, the user has default read access to the CIM objects within the specified namespace.

You can restrict access by all users to a namespace, and then grant individual users read, read-write, or write access to that namespace.

You cannot set access rights to individual managed objects. However, you can set access rights for all managed objects within a namespace and on a per-user basis.

If you log in as root, you can use the WBEM User Manager to set the following types of access to CIM objects:

- Read Only — Allows read-only access to objects within the CIM schema. Users with Read Only privileges can retrieve instances and classes, but cannot create, delete, nor modify CIM objects. The default user access.
- 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.

▼ To Start the Sun WBEM User Manager

1. **Enter the following command on the command line as root:**

   ```
   # /usr/sadm/bin/wbemadmin
   ```

   The Sun WBEM User Manager is loaded, and the Login dialog is displayed. To use context-sensitive help, click on fields in the dialog to display the Context Help panel.

2. **In the Login dialog, enter the user name in the User Name field.**

   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 not change, user privileges.

   To grant access rights to users, you must log in either as root or as a user with Write access to the `root\security` namespace.

3. **In the Login dialog, enter the password for the user account in the Password field.**
4. **Click OK.**

   The User Manager dialog is displayed. It contains a list of users and their access rights to WBEM objects within the namespaces on the current domain.

▼ To Grant Default Access Rights to a User

1. **Start the Sun WBEM User Manager.**
2. **Click Add in the Users Access portion of the User Manager dialog.**

   A dialog is displayed that lists all available namespaces on the domain.
3. **Type the Solaris user's account name in the User Name field.**
4. **Select a namespace from the list of available namespaces.**
5. **Click OK.**

   The user name is added to the list of users shown in the User Manager dialog.
6. **Click OK to save the changes and close the User Manager dialog. Or, click Apply to save the changes and leave the dialog open.**

   The user now has Read Only access to CIM objects in the selected namespaces.

▼ To Change a User's Access Rights

1. **Start the Sun WBEM User Manager.**
2. **Select the user from the list whose access rights you want to change.**
3. **To grant Read Only access to the user, 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. Or, click Apply to save the changes and leave the dialog open.**

▼ To Remove a User's Access Rights

1. **Start the Sun WBEM User Manager.**
2. **In the Users Access portion of the User Manager dialog, select the user from the list whose access rights you want to remove.**

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

   A confirmation dialog prompts you to confirm that you want to revoke the user's access rights. Click OK to proceed.

4. **Click OK to save the changes and close the User Manager dialog. Or, click Apply to save the changes and leave the dialog open.**

▼ To Set Access Rights for a Namespace

1. **Start the Sun WBEM User Manager.**
2. **In the Namespace Access portion of the User Manager dialog, click Add.**

   A dialog is displayed that lists all the namespaces that are available in the domain.

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

   By default users have Read Only access to namespaces, and the Read check box is checked. To allow Write access, click the Write check box. To allow Read/Write access click both the Read and Write check boxes. To allow no access to the namespace, make sure both the Read and Write check boxes are not checked.

4. **Click OK to save the changes and close the User Manager dialog. Or, click Apply to save the changes and leave the dialog open.**

▼ To Remove Access Rights for a Namespace

1. **Start the Sun WBEM User Manager.**
2. **In the Namespace Access portion of the User Manager dialog, select the namespace whose access rights you want to remove and click Delete.**

   This removes access control from the namespace, and removes the namespace from the list of namespaces displayed in the User Manager dialog box.

3. **Click OK to save the changes and close the User Manager dialog. Or, click Apply to save the changes and leave the dialog open.**

Using 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` that has a default value of "`r`" (Read Only).

You can set access privileges by setting the `capability` property of the `Solaris_UserAcl` class to one of the following values:

TABLE 2-2 Settings of the `capability` Property

Access Right	Description
`r`	Read Only
`rw`	Read / Write
`w`	Write
`none`	Only

In addition to the `capability` property, the `Solaris_UserAcl` class defines the following two key properties. Only one instance of the `namespace-username` ACL pair can exist in a namespace.

TABLE 2-3 Key Properties of the `Solaris_UserAcl` class

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

▼ Setting Access Control on a User

1. **Create an instance of the Solaris_UserAcl class, using code such as the following:**

```
...
/* 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, using code such as the following:**

```
...
/* 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 newly created instance using code such as the following:**

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

The `Solaris_NamespaceAcl` Class

The `Solaris_NamespaceAcl` class extends the `Solaris_Acl` base class, from which it inherits the string property `capability` whose default value is "`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 ACL applies. Only one instance of the namespace ACL can exist in a namespace.

▼ Setting Access Control on a Namespace

1. **Create an instance of the Solaris_namespaceACL class, using code such as the following:**

```
...
/* 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 grant the desired access rights, using code such as the following:**

```
...
/* 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();
...
```

3. **Update the newly created instance, using code such as the following:**

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

Solaris Management Console (SMC) Users Tool

The SMC Users tool lets you add users to existing roles and grant RBAC rights to existing users. RBAC rights are managed in the Rights portion of the SMC Users tool.

▼ To Start SMC and the Users Tool

1. **Enter the following command to change to the location of the SMC invocation command:**

```
# cd /usr/sbin
```

2. **Type the following command to start the SMC:**

```
# smc
```

3. **After the application is loaded and the user interface is displayed, double-click "This Computer" (or single-click the expand/compress icon next to "This Computer") in the left-hand Navigation panel to expand the tree beneath "This Computer."**
4. **Double-click "System Configuration" (or single-click the expand/compress icon next to "System Configuration") in the left-hand Navigation panel to expand the tree beneath "System Configuration." The Users icon is displayed.**
5. **Click the Users icon to start the Users Tool.**

Note – For more information about using the Solaris Management Console, see the `smc`(1m) man page.

Solaris WBEM Logging Services

WBEM Logging services enable systems administrators to monitor system events and to determine how they occurred.

About Solaris WBEM 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. In addition, 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 the section "Using the APIs to Enable Solaris WBEM Logging" on page 32.

You can view log data in the Solaris Management Console (SMC) Log Viewer to debug the logging functionality that you have set up. For more information about viewing log files, see the section "Solaris WBEM Log Viewer" on page 39, and the `smc`(1m) man page.

Solaris WBEM Services 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: `/var/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`" is the most recently-saved version, such as `wbem_log.1`. A log file appended with a "`.2`" is the next oldest version, and so on. All versions of the log file co-exist as an archive in `/var/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 the section "Solaris WBEM Services Log File Rules" on page 29.

- 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 the section "Solaris_LogService Class" on page 31.

Solaris WBEM Services 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:

- The directory where the log file is written
- The name of the log file
- The size allowed for a log file before it is renamed with a `.1` file extension and saved.
- The number of log files you can have in the archive
- The 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 the `Solaris_LogServiceProperties` class and set the values of its associated properties. See the section “Setting Solaris WBEM Logging Properties” on page 38 for detailed information about how to set property values of the new instance.

Solaris WBEM Services 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 is 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` — The type of log record
- `Severity` — The severity of conditions that caused data to be written to a log file
- `AppName` — The name of the application from which the data was obtained
- `UserName` — The name of the individual who was using the application when log data was generated
- `ClientMachineName` — The name of the computer on which an incident occurred that generated log data.
- `ServerMachineName` —- The name of the server on which an incident occurred that generated log data
- `SummaryMessage` — A brief message describing the occurrence
- `DetailedMessage` — A detailed message describing the occurrence
- `Data` — Context information that applications and providers can present to interpret a log message.

Solaris WBEM Log Classes

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

Solaris_LogRecord Class

The Solaris_LogRecord class is defined in the Solaris_Core1.0.mof file 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. The Solaris_Core1.0.mof file is located in /usr/sadm/mof.

The Solaris_LogRecord class 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 Class

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.

The `Solaris_LogServiceProperties` class lets you set logging properties. See the section “Setting Solaris WBEM Logging Properties” on page 38.

You can view the definition of the `Solaris_LogService` class in the `Solaris_Core1.0.mof` file, which is located in `/usr/sadm/mof`.

Using the APIs to Enable Solaris WBEM 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 Solaris WBEM 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 the necessary Java classes. The minimum classes are:**

```
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 create instances of the following classes: `CIMClient`, `CIMObjectPath`, and `CIMNameSpace`:**

```
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.**

```
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 class for the log record.**

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

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

```
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.**

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

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

Reading Data from a Solaris WBEM 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 for the log viewer

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

1. **.Import all the necessary Java classes. The classes listed below are the minimum required:**

```
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`.**

```
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 the `CIMClient`, `CIMObjectPath`, and `CIMNameSpace` values of the `ReadLog` class.**

```
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 the instances of `Solaris_LogRecord`.**

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

5. **Send the property values to an output device.**

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

   (((Long)ci.getProperty("RecordID").getValue().getValue()).longVal
   ue()));
System.out.println("Log filename : " +
   ((String)ci.getProperty("FileName").getValue().getValue()));
int categ = 0
   (((Integer)ci.getProperty("category").getValue().getValue()).intV
   alue());
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()).intV
   alue());
```

```
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.

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

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

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

Setting Solaris WBEM 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 code example shows how to set logging properties. Properties are stored in the `/var/sadm/lib/wbem/WbemServices.properties` file.

```
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]);
   CIMObjectPath logpropcop = new
   CIMObjectPath("Solaris_LogServiceProperties");
   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();
   }
}
```

Solaris WBEM Log Viewer

You can view all details of a log record in the Solaris Management Console (SMC) Log Viewer, an application that provides a graphical user interface for viewing recorded data. For more information on the SMC, see the man page `smc`(1M).

After you have created a log record, you can start the SMC and then its Log Viewer.

▼ How to Start SMC and Solaris Log Viewer

1. **Change to the location of the SMC invocation command by typing the following:**

   ```
   # cd /usr/sbin
   ```

2. **Start SMC by typing the following command:**

   ```
   # smc
   ```

3. **In the Navigation panel, double-click This Computer (or single-click the expand/compress icon next to it) to expand the tree beneath it. Double-click System Status and the Log Viewer icon will be displayed.**

4. **Click the Log Viewer icon to start the application.**

CHAPTER 3

Using Process Indications

This chapter describes CIM process indications; how they are used to communicate the occurrence of events; and the classes that enable clients to subscribe to receive CIM process indications. This chapter includes the following topics:

For more information about process indication classes, see Chapter 4, "Classes, Domains, Associations, and Indications in WDR."

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

The CIM Event Model

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

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

A process indication is a subtype of a class that has an association with zero or more *triggers* (descriptions of changes in data that result from events) that can create instances of the class `Indication`. The WBEM implementation does not have an explicitly defined object that represents a trigger. Triggers are implied either by the operations on basic objects of the system (`create`, `delete`, and `modify` on classes, instances, and namespaces) or by events in the managed environment. When an event takes place, the WBEM provider generates a process indication that something happened in the system.

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

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

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

The `CIM_ProcessIndication` superclass resides at the top of the "The WDR Indication Class Hierarchy Diagram" on page 83.

How Indications are Generated

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

Administrators can change the event polling interval and the default polling behavior of the CIM Object Manager by editing the properties in the `cimom.properties` file. For instructions on editing the `cimom.properties` file, see the *Solaris WBEM Services Administrator's Guide* (part number 806-6468-10).

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

How Subscriptions Are Created

A client application can subscribe to be notified of CIM events. A *subscription* is a declaration of interest in one or more streams of indications.

An application that subscribes for indications of CIM events describes:

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

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

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

`CIM_IndicationFilter` — Defines the criteria for generating an indication and which data should be returned in the indication.

`CIM_IndicationHandler` — Describes how to process and handle an indication. May include a destination and a protocol for delivering indications.

`CIM_IndicationSubscription` — An association that binds an event filter with an event handler.

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

Adding a CIM Listener

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

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

Adding a CIM Listener

Use code such as the following to add a CIM listener:

```
// Connect to the CIM Object Manager
cc = new CIMClient();
// Register the CIM Listener
cc.addCIMListener(new CIMListener() {
   public void indicationOccured(CIMEvent e) {
   }
});
```

Creating an Event Filter

Event filters describe the types of events to be delivered and the conditions under which they are delivered. An application creates an event filter by creating an instance of the `CIM_IndicationFilter` class and defining values for its properties. Event filters belong to a namespace. Each event filter works only on events that belong to the namespace to which the filter also belongs.

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

TABLE 3-1 Properties in the CIM_IndicationFilter Class

Property	Description	Required/Optional
SystemCreationClassName	The name of the system on which the creation class for the filter resides, or to which it applies	Optional. The default for this key property is the CIMSystem.CreationClassName
SystemName	The name of the system on which the filter resides, or to which it applies	Optional. The default for this key property is the name of the system on which the CIM Object Manager is running.
CreationClassName	The name of the class or subclass that was used to create the filter	Optional. The CIM Object Manager assigns CIM_IndicationFilter as the default for this key property.
Name	The unique name of the filter	Optional. The CIM Object Manager assigns a unique name.
SourceNamespace	The path to a local namespace where the CIM indications originate	Optional. The default is null.
Query	A query expression that defines the conditions under which indications are generated. Currently, only Level 1 WBEM Query Language expressions are supported. To learn how to construct WQL query expressions, see the section "Querying" in the *Sun WBEM SDK Developer's Guide* (part number 806-6831-10).	Required
QueryLanguage	The language in which the query expression is written.	Required. The default is WQL (WBEM Query Language).

▼ To Create an Event Filter

1. **Create an instance of the `CIM_IndicationFilter` Class, using code such as the following:**

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

2. **Specify the name of the event filter, using code such as the following:**

```
Name = ''filter_all_new_solarisdiskdrives'';
```

3. **Create a WQL string to identify event indications to be returned, using code such as the following:**

```
String filterString = ''SELECT *
  FROM CIM_InstCreation WHERE sourceInstance is
  ISA Solaris_DiskDrive''
```

4. **Set property values in the `cimfilter` instance to identify the name of the filter, the filter string that selects CIM events, and the query language used to parse the query string, using code such as the following.**

Note – Currently, only the WBEM Query Language can be used to parse query strings.

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

5. **Create an instance from the `cimfilter` instance and store it in the CIM Object Manager Repository, using code such as the following:**

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

Creating an Event Handler

The Solaris Event MOF extends the `CIM_IndicationHandler` class by creating the `Solaris_JAVARXMIDelivery` class to handle delivery of indications of CIM events to client applications using the RMI protocol. RMI clients must instantiate the `Solaris_JAVAXRMIDelivery` class to set up an RMI delivery location. Clients can use only RMI to receive events; HTTP is not supported.

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

TABLE 3-2 Properties in the `CIM_IndicationHandler` Class

Property	Description	Required/Optional
`SystemCreationClassName`	The name of the system on which the creation class for the handler resides, or to which it applies	Optional. Set by the CIM Object Manager.
`SystemName`	The name of the system on which the handler resides, or to which it applies	Optional. The default for this key property is the name of the system on which the CIM Object Manager is running.
`CreationClassName`	The name of the class or subclass that was used to create the handler	Optional. The CIM Object Manager assigns the appropriate class as the default for this key property.
`Name`	The unique name of the handler	Required. The client application must assign a unique name.
`Owner`	The name of the entity that created, or that maintains, this handler. The provider can check this value to determine whether to authorize a handler to receive an indication.	Optional. The default value is the Solaris user name of the user who is creating the instance.

▼ Creating a CIM Event Handler

To create a CIM event handler, use code such as the following:

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

CIMInstance ci = rmidelivery.newInstance();

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

Binding an Event Filter to an Event Handler

An application binds an event filter to an event handler by creating an instance of the `CIM_IndicationSubscription` class. When a `CIM_IndicationSubscription` is created, indications for the events specified by the event filter are delivered.

The following example code creates a subscription (`filterdelivery`) and defines the `filter` property to the `filter` object that was created in “Creating an Event Filter” on page 44, and defines the `handler` property to the `delivery` object created in “Creating a CIM Event Handler” on page 48:

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

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

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

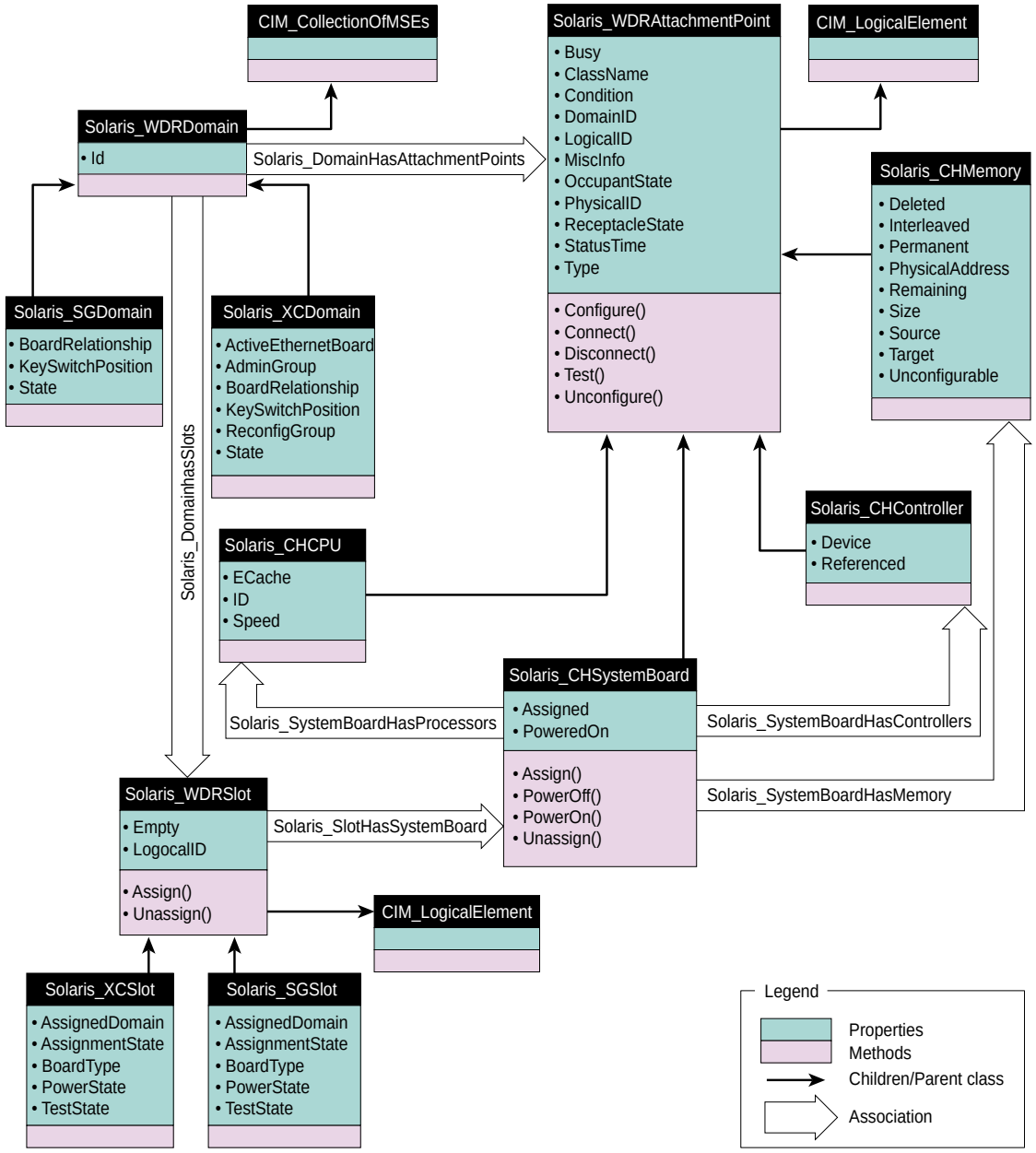
```
CIMObjectPath indsub = cc.createInstance(new CIMObjectPath(),
ci);
```

CHAPTER 4

Classes, Domains, Associations, and Indications in WDR

This chapter contains the following five sections:

WDR CIM Class Hierarchy Diagram



CIM Attachment Point Classes

Attachment point classes provide logical elements that represent attachment points in Sun Fire 15K, 12K, 6800, 4810, 4800, or 3800 systems. An attachment point is an interface to a physical location in Sun Fire 15K, 12K, 6800, 4810, 4800, or 3800 systems where you can use WDR to configure system boards, CPUs, and memory modules in domains that are running the Solaris operating environment. An attachment point is comprised of a receptacle and an occupant. When you insert an occupant into a receptacle or remove it from a receptacle, the attachment point's state changes.

Note – For more information about attachment points, refer to the `cfgadm(1M)` man page (all Sun Fire models) and the `cfgadm_sbd`(1M) man page (Sun Fire 15K and 12K only).

Attachment point classes are similar to Slot classes insofar as they represent physical locations in Sun Fire 15K, 12K, 6800, 4810, 4800, or 3800 systems where you can use WDR. (See the section "CIM Slot Classes" on page 64.) However, Slot classes provide logical elements that represent *only* system board and I/O boards, and not CPUs, memory, and I/O controllers. Slots are a type of attachment point whose scope is limited only to boards.

CIM Solaris_WDRAttachmentPoint Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRAttachmentPoint
```

Description

Represents the core Configuration Administration (`cfgadm`) information. This information is gathered using the `libcfgadm` library on domains.

Direct Known Subclasses

CIM Solaris_CHCPU Class, CIM Solaris_CHSystemBoard Class, CIM Solaris_CHController Class, and CIM Solaris_CHMemory Class

CIM Solaris_WDRAttachmentPoint Class Properties

Note – For more information about attachment points, refer to the `cfgadm`(1M) man page (all Sun Fire systems), and the `cfgadm_sbd`(1M) man page (Sun Fire 15K and 12K only).

TABLE 4-1 CIM Solaris_WDRAttachmentPoint Properties

Property	Data Type	Description
`ClassName`	string	The class of attachment point. For example, “sbd” represents a system board.
`Busy`	uint32	Indicates whether the attachment point is currently in a state transition.
`Condition`	uint32	The condition of the attachment point. Possible values: `Unknown`, `OK`, `Failing`, `Failed`, and `Unusable`
`LogicalID`	string	The logical identifier of the attachment point
`PhysicalID`	string	The physical identifier of the attachment point. For example: `/devices/pseudo/dr@0::SB6`
`DomainID`	uint32	The domain to which this attachment point is assigned or available. On Sun Fire 15K systems, domains are numbered between 0 and 17. On Sun Fire 12K systems, domains are numbered between 0 and 8. On Sun Fire 3800, 4800, and 4810 systems, domains are numbered 0 and 1 (maximum two domains). On Sun Fire 6800 systems, domains are numbered between 0 and 3 (maximum four domains).
`OccupantState`	uint32	The occupant state of the attachment point. Possible values: `None`, `Configured`, and `Unconfigured`
`ReceptacleState`	uint32	The receptacle state of the attachment point. Possible values: `None`, `Empty`, `Disconnected`, and `Connected`
`Type`	string	The type of the attachment point. Either cpu*n*, pci*n*, or mem*n*, where *n* is the number of the component.

TABLE 4-1 CIM Solaris_WDRAttachmentPoint Properties

`MiscInfo`	string	Driver-specific information that the driver sets. A list of name-value pairs. Depends on the value of the `Type` property. For example, if the `Type` property is cpu*n*, the `MiscInfo` property contains is populated with the following information: the Processor ID, the Processor speed, and the Ecache memory size in MB.
`StatusTime`	datetime	The date and time of the latest status change to the attachment point, in the following format: `yyyymmddhhmmss.mmmmmmsutc` Where: `yyyy` represents the year, `mm` represents the month, `dd` represents the day, `hh` represents the hour, `mm` represents the minutes, `ss` represents the seconds,

CIM Solaris_WDRAttachmentPoint Class Methods

TABLE 4-2 CIM Solaris_WDRAttachmentPoint Methods

Name	Return Value	Description
`Configure`	sint32	Configures the attachment point into a Solaris domain. Has the following parameters: `force` — boolean `hardwareOpts` — string `retries` — uint32 retries `retryDelay` — uint32 Returns the following: `error` — string
`Unconfigure`	sint32	Removes the resources of the attachment point from the Solaris domain in which it is currently configured. Has the following parameters: `force` — boolean `hardwareOpts` — string `retries` — uint32 retries `retryDelay` — uint32 Returns the following: `error` — string
`Connect`	sint32	Changes the receptacle state to connected. Has the following parameters: `force` — boolean `hardwareOpts` — string `retries` — uint32 retries `retryDelay` — uint32 Returns the following: `error` — string

TABLE 4-2 CIM Solaris_WDRAttachmentPoint Methods

`Disconnect`	sint32	Disables normal communication to or from the occupant in a receptacle. Has the following parameters: `force` — boolean `hardwareOpts` — string `retries` — uint32 retries `retryDelay` — uint32 Returns the following: `error` — string
`Test`	sint32	Tests the condition of the attachment point. Has the following parameters: verbose — boolean `hardwareOpts` — string Returns the following: `error` — string

CIM Solaris_CHSystemBoard Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRAttachmentPoint
      |
      +--Solaris_CHSystemBoard
```

Description

Represents a logical element that models the UltraSPARC-III generation of system boards that support the functionality of Dynamic Reconfiguration Model 2.0.

As illustrated in the “WDR CIM Class Hierarchy Diagram” on page 52, the CIM `Solaris_CHSystemBoard` class has association relationships with the following CIM classes: `Solaris_CHMemory`, `Solaris_CHController`, `Solaris_WDRSlot` and `Solaris_CHCPU`.

Direct Known Subclasses

None

CIM Solaris_CHSystemBoard Class Properties

TABLE 4-3 CIM Solaris_CHSystemBoard Properties

Name	Data Type	Description
`Assigned`	boolean	Indicates that the board is assigned to a Solaris domain.
`PoweredOn`	boolean	Indicates that the board is powered-on.

CIM Solaris_CHSystemBoard Class Methods

TABLE 4-4 CIM Solaris_CH_SystemBoard Methods

Name	Return Value	Description
`Assign`	sint32	Assigns the board to a specified Solaris domain. Has the following parameters: `force` — boolean `hardwareOpts` — string Returns the following: `error` — string
`PowerOn`	sint32	Powers-on the board. Has the following parameters: `force` — boolean `hardwareOpts` — string Returns the following: `error` — string
`PowerOff`	sint32	Powers-off the board. Has the following parameters: `force` — boolean `hardwareOpts` — string Returns the following: `error` — string

TABLE 4-4 CIM Solaris_CH_SystemBoard Methods

`Unassign`	sint32	Unassigns the board from the domain to which it is currently assigned. Has the following parameters: `force` — boolean `hardwareOpts` — string Returns the following: `error` — string

CIM Solaris_CHCPU Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRAttachmentPoint
      |
      +--Solaris_CHCPU
```

Description

A logical element that represents a processor on a system board. There can be as many as four processors per system board on an UltraSPARC-III generation system board. Because the processor is physically attached to a CPU socket on a system board, and because DR operations such as configure and unconfigure can be performed on the attachment point, the CIM `Solaris_CHCPU` class is derived from the CIM `Solaris_WDRAttachmentPoint` class.

As illustrated in the “WDR CIM Class Hierarchy Diagram” on page 52, the CIM `Solaris_CHCPU` class has an aggregation relationship with the CIM `Solaris_CHSystemBoard` class.

Direct Known Subclasses

None

CIM Solaris_CHCPU Class Properties

TABLE 4-5 Solaris_CHCPU Properties

Name	Data Type	Description
`ID`	uint32	A unique identifier for the processor
`Speed`	uint32	The clock speed of the processor in MHz
`ECache`	uint32	The size of the ECache memory in MB.

CIM Solaris_CHCPU Class Methods

None

CIM Solaris_CHMemory Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRAttachmentPoint
      |
      +--Solaris_CHMemory
```

Description

A logical element that describes the memory information for a system board. There is a one-to-one relationship between instances of the `Solaris_CHSystemBoard` and `Solaris_CHMemory` CIM classes. Furthermore, because memory is an attachment point on the system board, the CIM `Solaris_CHMemory` class is derived from the CIM `Solaris_WDRAttachmentPoint` class.

Direct Known Subclasses

None

CIM Solaris_CHMemory Properties

TABLE 4-6 CIM Solaris_CHMemory Properties

Name	Data Type	Description
`Deleted`	uint32	While a memory drain is in progress, the Deleted property stores the amount of memory that has already been deleted. Otherwise the `Deleted` property is null.
`Interleaved`	boolean	True if the board is participating in interleaving with other boards.
`Permanent`	uint32	Stores the number of non-pageable memory pages in the board's memory, in kilobytes.
`PhysicalAddress`	uint64	The base physical address of memory on the board
`Remaining`	uint32	When a memory drain is in progress, the Remaining property stores the amount of remaining memory that needs to be drained, in megabytes. Otherwise the `Remaining` property is null.
`Size`	uint32	The size of memory on the board in megabytes
`Source`	string	The name of the copy-rename source attachment point. When there is no copy-rename operation, the `Source` property is null.
`Target`	string	The name of the copy-rename target attachment point. When there is no copy-rename operation, the `Target` property is null.
`Unconfigurable`	boolean	True if the operating system has been configured to disallow this memory from being unconfigured.

CIM Solaris_CHMemory Class Methods

None

CIM Solaris_CHController Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRAttachmentPoint
      |
      +--Solaris_CHController
```

Description

A logical CIM element that models the I/O controller attachment points on an I/O board.

Direct Known Subclasses

None

CIM Solaris_CHController Class Properties

TABLE 4-7 Solaris_CHController Properties

Name	Data Type	Description
`Device`	string	The physical path of the I/O component in the `/devices` path
`Referenced`	boolean	True if the I/O component is referenced.

CIM Solaris_CHController Class Methods

None

CIM Slot Classes

The CIM Slot classes model system board slots on Sun Fire 15K, 12K, 3800, 4800, 4810, and 6800 systems. The slots can be empty or occupied. Like attachment points, slots can be assigned to, and unassigned from, domains. However, unlike attachment points, slots can exist independent of any domain, and they always exist.

Note – Classes whose names contain “XC” are used with Sun Fire™ 15K and 12K systems. Classes whose names contain “SG” are used with Sun Fire 6800, 4810, 4800, and 3800 systems.

CIM Solaris_WDRSlot Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRSlot
```

The abstract CIM `Solaris_WDRSlot` class models a platform-independent slot.

Description

A logical CIM element that provides a superclass to those logical CIM elements that model the slots in a Sun Fire 15K, 12K, 6800, 4810, 4800, or 3800 chassis. A slot can contain either a system board or an I/O board.

As illustrated in the “WDR CIM Class Hierarchy Diagram” on page 52, the `Solaris_WDRSlot` class has association relationships with the following CIM classes: `Solaris_CHSystemBoard` and `Solaris_WDRDomain`.

Direct Known Subclasses

`CIM Solaris_XCSlot Class` and `CIM Solaris_SGSlot Class`

CIM Solaris_WDRSlot Properties

TABLE 4-8 CIM Solaris_WDRSlot Properties

Name	Data Type	Description
`LogicalID`	string	The logical name of the slot. On a Sun Fire 15K system there are 18 expanders, and each can hold one system board and one I/O board. System board slots are represented as SB0, SB1, ... SB17, and I/O board slots are represented as IO0, IO1, ... IO17. On a Sun Fire 12K system there are 9 expanders, and each can hold one system board and one I/O board. System board slots are represented as SB0, SB1, ... SB8, and I/O board slots are represented as IO0, IO1, ... IO8. On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.
`Empty`	boolean	Indicates whether this slot contains a board. A value of NULL indicates that the state of the slot is unknown. If the `Empty` property is True, then the following properties of the `CIM Solaris_XCSlot Class` and the `CIM Solaris_SGSlot Class` are NULL: `AssignmentState`, `BoardType`, `PowerState`, and `TestState`.

CIM Solaris_WDRSlot Methods

TABLE 4-9 CIM Solaris_WDRSlot Methods

Name	Return Value	Description
`Assign`	sint32	Assigns the slot to the specified domain. Has the following parameter: `Assign` — uint32 Returns the following value: `error` — string

TABLE 4-9 CIM Solaris_WDRSlot Methods

`Unassign`	sint32	Unassigns a board from a domain. No board in the slot can be active (i.e., connected or configured) in the domain. Has the following parameter: `Assign` — uint32 Returns the following value: `error` — string

CIM Solaris_XCSlot Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRSlot
      |
      +--Solaris_XCSlot
```

Description

A logical CIM element that models the slots on a Sun Fire 15K or 12K system. A slot can contain either a system board or an I/O board.

On a Sun Fire 15K system there are 18 expanders, and each can hold one system board and one I/O board. System board slots are represented as SB0, SB1, ... SB17, and I/O board slots are represented as IO0 (zero), IO1, IO2, ... IO17.

On a Sun Fire 12K system there are 9 expanders, and each can hold one system board and one I/O board. System board slots are represented as SB0, SB1, ... SB8, and I/O board slots are represented as IO0 (zero), IO1, IO2, ... IO8.

Direct Known Subclasses

None

CIM Solaris_XCSlot Properties

TABLE 4-10 CIM Solaris_XCSlot Properties

Name	Data Type	Description
AssignedDomain	sint32	The domain to which this slot is assigned, if the value of its AssignmentState property is Assigned. The numeric Values -1 through 18 represent the following in the ValueMap:: None, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, N, P, Q, and R.
AssignmentState	uint32	The current assignment state of the slot. The Values 0 through 3 represent the following in the ValueMap: Unknown, Free, Assigned, and Active. Always NULL is the Empty property (inherited from the Solaris_WDRSlot class) is True.
BoardType	uint32	The type of board that resides in the slot, if known. The Values 0 through 8 represent the following items in the ValueMap: CPU, WIB, HPCI, CPCI, MCPU, WPCI, SPCI, HPCIX, and Unknown. Note: Unknown is not equal to Empty. Always NULL is the Empty property (inherited from the Solaris_WDRSlot class) is True.
PowerState	uint32	The power state of the board. The Values 0 through 3 represent the following items in the ValueMap: Off, On, Unknown, or Minimal. Always NULL is the Empty property (inherited from the Solaris_WDRSlot class) is True.
TestState	uint32	The test state of the board. The numeric Values 0 through 4 represent the following in the ValueMap: Unknown, iPOST, Passed, Degraded, or Failed. Always NULL is the Empty property (inherited from the Solaris_WDRSlot class) is True.

CIM Solaris_XCSlot Methods

None

CIM Solaris_SGSlot Class

Position in the Class Hierarchy

```
CIM_LogicalElement
   |
   +--Solaris_WDRSlot
      |
      +--Solaris_SGSlot
```

Description

A logical CIM element that models the slots on a Sun Fire 6800, 4810, 4800, or 3800 system.

Note – On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.

Direct Known Subclasses

None

CIM Solaris_SGSlot Properties

TABLE 4-11 CIM Solaris_SGSlot Properties

Name	Data Type	Description
`AssignedDomain`	sint32	The domain to which this slot is assigned, if the value of the slot's `AssignmentState` property is `Assigned`. The Values 1 through 5 represent the following items in the ValueMap: • None • A • B • C • D
`AssignmentState`	uint32	The current assignment state of the slot. The Values 1 through 4 represent the following in the ValueMap: • Unknown • Free • Assigned • Active
`BoardType`	uint32	The type of board that resides in the slot if known. The Values 1 through 11 represent the following items in the ValueMap: • Unknown • Empty • CPU • IO • CPUWIB • IOWIB • SC • L2 • Fan • Power Supply • Logic Analyzer
`PowerState`	uint32	The power state of the board. The Values 1 through 4 represent the following items in the ValueMap: • Unknown • On • Off • Failed

TABLE 4-11 CIM Solaris_SGSlot Properties

`TestState`	uint32	The test state of the board. The Values 1 through 8 represent the following items in the ValueMap: • Unknown • Not Tested • Passed • Failed • Under Test • Start Test • Degraded • Unusable

CIM Solaris_SGSlot Methods

None

CIM Solaris_WDRDomain Classes

The CIM Solaris domain classes represent domains on Sun Fire systems that are running the Solaris operating environment.

CIM Solaris_WDRDomain Class

Description

The CIM `Solaris_WDRDomain` class is an abstract superclass that describes domain information on all Sun Fire systems (the 15K, 12K 6800, 4810, 4800, and 3800 systems).

As illustrated in the“WDR CIM Class Hierarchy Diagram” on page 52, the CIM `Solaris_WDRDomain` class has an association relationship with the `Solaris_WDRSlot` class and an aggregation relationship with the `Solaris_WDRAttachmentPoint` class.

Position in the Class Hierarchy

```
CIM_CollectionOfMSEs
   |
   +--Solaris_WDRDomain
```

Direct Known CIM Subclasses

`CIM Solaris_SGDomain Class` and `CIM Solaris_XCDomain Class`

Note – CIM domain classes whose names contain “XC” are used with Sun Fire™ 15K and 12K systems. CIM domain classes whose names contain “SG” are used with Sun Fire 6800, 4810, 4800, and 3800 systems.

CIM Solaris_WDRDomain Class Properties

TABLE 4-12 CIM Solaris_WDRDomain Properties

Name	Data Type	Description
`Id`	uint32	Identifies the domain uniquely.

CIM Solaris_XCDomain Class

Description

The CIM `Solaris_XCDomain` class, which is a subclass of the `CIM Solaris_WDRDomain` class, describes domain information on Sun Fire 15K and 12K systems. It contains several CIM properties that contain information that is specific to Sun Fire 15K and 12K systems.

Position in the Class Hierarchy

```
CIM_CollectionOfMSEs
   |
   +--Solaris_WDRDomain
      |
      +--Solaris_XCDomain
```

Direct Known CIM Subclasses

None

CIM Solaris_XCDomain Class Properties

TABLE 4-13 CIM Solaris_XCDomain Properties

Name	Data Type	Description
`ActiveEthernetBoard`	string	The I/O board that hosts the active Ethernet connection for the internal system controller (SC) network.
`AdminGroup`	string	The name of the UNIX group that is assigned to the Domain Administrator group
`BoardRelationship[]`	sint32	An array of values, one for each board, that indicates the status of the board within the domain. Each position in the array's BitMap represents the status of one board; each number in the ValueMap represents one of the following Values: • Not Available • Available • Assigned • Active Numbers 1 through 18 in the array's BitMap represent the status of each system board (SB0 through SB17). Numbers 19 through 36 in the array's BitMap represent the status of each I/O board (IO0 through IO17).
`KeyswitchPosition`	uint32	Indicates the status of the domain. Each of the Values 0 through 5 represents an item in the ValueMap, which indicates the status of the domain: • On • Standby • Off • Diag • Secure • Unknown
`ReconfigGroup`	string	The name of the UNIX group that is assigned to the Domain Reconfiguration role.

TABLE 4-13 CIM Solaris_XCDomain Properties

`State`	uint32	The current state of the domain. Each number, 0 through 36, in the ValueMap represents one of the following Values, which indicate the current state of the domain: • Unknown • Powered Off • Keyswitch Standby • Running Domain POST • Running Board POST • Layout OBP • Loading OBP • OBP Booting • OBP Running • OBP Callback • OBP Loading Solaris • OBP Booting Solaris • OBP Domain Exited • OBP Failed • OBP in Sync Callback • OBP Exited • OBP Error Reset • OBP Domain Halt • OBP Environmental Domain Halt • OBP Booting Solaris Failed • OBP Loading Solaris Failed • OBP Debug • OS Running Solaris • OS Quiesce in Progress • OS Quiesced • OS Resume in Progress • OS Panic • OS Panic Debug • OS Panic Continue • OS Panic Dump • OS Halt • OS Panic Exit • OS Environmental Exit • OS Debug • OS Exit • Domain Down • Domain In Recovery

CIM Solaris_SGDomain Class

Description

The CIM `Solaris_SGDomain` class, which is a subclass of the `CIM Solaris_WDRDomain` class, describes domain information on Sun Fire 6800, 4810, 4800, and 3800 systems. It contains several CIM properties that contain information that is specific to Sun Fire 6800, 4810, 4800, and 3800 systems.

Position in the Class Hierarchy

```
CIM_CollectionOfMSEs
   |
   +--Solaris_WDRDomain
      |
      +--Solaris_SGDomain
```

Direct Known CIM Subclasses

None

CIM Solaris_SGDomain Class Properties

TABLE 4-14 CIM Solaris_SGDomain Properties

Name	Data Type	Description
`BoardRelationship[]`	sint32	An array of values, one for each board, that indicates the status of the board in the domain. For each position in the array BitMap, ValueMap items 0 through 4 represents the following board status values: • Nonexistent Slot • Not Available • Available • Assigned • Active On a Sun Fire 6800 system, the BitMap values 1 through 10 represent all boards. BitMap values 1 through 6 relate to system boards 0 through 5 (SB0 through SB5). BitMap values 7 through 10 relate to I/O boards, IB6 through IB9. On Sun Fire 4810, 4800, and 3800 systems, only five slots are available, for three CPU boards and two I/O boards. Therefore, the BitMap values 4, 5, and 6 (for SB3, SB4, and SB5), and BitMap values 9 and 10 (for IB8 and IB9), are always 0 (Nonexistent Slot).
`KeyswitchPosition`	uint32	Indicates the status of the domain. The Values 1 through 16 represent the following items in the ValueMap: • Unknown • Off • Standby • On • Diag • Secure • Off To Standby • Off To On • Off To Diag • Off To Secure • Standby To Off • Active To Off • Active To Standby • Reboot To On • Reboot To Diag • Reboot To Secure

TABLE 4-14 CIM Solaris_SGDomain Properties

State	uint32	The current state of the domain. The ValueMap items 1 through 14 represent the following values: • Unknown • Running POST • Standby • Active • Powered Off • Domain Idle • Running OBP • Booting • Running Solaris • Halted • Reset • Panic • Debugger • Hang Detected

WDR Schema Associations and Aggregations

A CIM association is a special class that relates one WDR class or instance to another. Associations can be one-to-one relationships or aggregations.

WDR aggregations relate one WDR class or instance to many other classes or instances.

CIM Solaris_DomainHasAttachmentPoints Aggregation

Description

A domain is said to have an attachment point if that attachment point is either available to the domain (and appears in the domain's ACL) or is assigned to the domain. Only domains that are running can have attachment points.

The Solaris_DomainHasAttachmentPoints aggregation relates sub-instances of the Solaris_WDRDomain class to the sub-instances of the Solaris_WDRAttachmentPoint class that are available or assigned to the domain.

The Solaris_DomainHasAttachmentPoints aggregation is a composition association where the domain is composed of one or more attachment points. The parent of the Solaris_DomainHasAttachmentPoints aggregation is a sun-instance of the Solaris_WDRDomain class. The child of the Solaris_DomainHasAttachmentPoints aggregation is a sub-instance of the Solaris_WDRAttachmentPoint class. The Solaris_DomainHasAttachmentPoints aggregation is a one-to-many relationship, where multiple attachment points can be available or assigned to a single domain.

CIM Solaris_DomainHasAttachmentPoints Aggregation Properties

TABLE 4-15 CIM Solaris_DomainHasAttachmentPoints Aggregation Properties

Name	Data Type	Description
Collection	REF Solaris_WDRDomain	References the parent in the relationship.
Member	REF Solaris_WDRAttachmentPoint	References a child in the relationship.

CIM Solaris_DomainHasSlots Aggregation

Description

One of the characteristics of a domain is that it contains zero or more slots. A slot can be assigned to a domain regardless of whether it is occupied by a system board. Consequently, the Solaris_DomainHasSlots aggregation relates the binding between the CIM Solaris_WDRDomain and CIM Solaris_WDRSlot classes.

The Solaris_DomainHasSlots aggregation is a composition association, where the domain is composed of one or more slots.

The parent of the Solaris_DomainHasSlots aggregation is an instance of the Solaris_XCDomain class, and the child is an instance of the Solaris_WDRSlot class. The Solaris_DomainHasSlots aggregation is a one-to-many relationship, where multiple slots can be assigned to a single domain. However, a single slot cannot reside in multiple domains at one time.

CIM Solaris_DomainHasSlots Aggregation Properties

TABLE 4-16 CIM Solaris_DomainHasSlots Aggregation Properties

Name	Data Type	Description
Collection	REF Solaris_WDRDomain	References the parent in the relationship.
Member	REF Solaris_WDRSlot	References a child in the relationship.

Solaris_SlotHasSystemBoard Association

Description

A slot can contain a board regardless of whether the slot is assigned to a domain. The CIM Solaris_SlotHasSystemBoard association relates an instance of the CIM Solaris_WDRSlot class to an instance of the CIM Solaris_SystemBoard class that corresponds to the board in the slot.

The CIM Solaris_SlotHasSystemBoard is a composition association, and an instance of the CIM Solaris_WDRSlot class can be composed of zero or one instance of the CIM Solaris_SystemBoard class.

CIM Solaris_SlotHasSystemBoard Association Properties

TABLE 4-17 CIM Solaris_SlotHasSystemBoard Association Properties

Name	Data Type	Description
Antecedent	REF Solaris_WDRSlot	References the parent in the relationship.
Dependent	REF Solaris_CHSystemBoard	References the child in the relationship.

Solaris_SystemBoardHasProcessors Aggregation

Description

A system board is a large circuit board that contains processors, a memory module, and I/O modules. The CIM `Solaris_SystemBoardHasProcessors` aggregation describes the relationship between an instance of the `Solaris_CHSystemBoard` class and an instance of the `Solaris_CHCPU` class; it relates a system board with the processors that it contains.

The aggregation is a one-to-many relationship where a board can contain between zero and four processors.

CIM Solaris_SystemBoardHasProcessors Aggregation Properties

TABLE 4-18 CIM Solaris_SystemBoardHasProcessors Aggregation Properties

Name	Data Type	Description
`GroupComponent`	REF Solaris_CHSystemBoard	References the parent in the relationship.
`PartComponent`	REF Solaris_CHCPU	References a child in the relationship.

Solaris_SystemBoardHasMemory Aggregation

Description

A system board is a large circuit board that contains processors, a memory module, and I/O modules. The CIM `Solaris_SystemBoardHasMemory` aggregation relates an instance of the `Solaris_CHSystemBoard` class with an instance of the `Solaris_CHMemory` class; it relates a board with the memory that it contains.

The `Solaris_CHMemory` class is a collection of information that describes memory on a system board. For a given system board, there is a maximum of one instance of the `Solaris_CHMemory` class.

CIM Solaris_SystemBoardHasMemory Aggregation Properties

TABLE 4-19 CIM Solaris_SystemBoardHasMemory Aggregation Properties

Name	Data Type	Description
`GroupComponent`	REF Solaris_CHSystemBoard	References the parent in the relationship.
`PartComponent`	REF Solaris_CHMemory	References a child in the relationship.

Solaris_SystemBoardHasControllers Aggregation

Description

In addition to processors and memory modules, a system board can have I/O modules such as disk and network controllers. The CIM `Solaris_SystemBoardHasControllers` aggregation relates a system board to the controllers that it contains.

`Solaris_SystemBoardHasControllers` is a one-to-many relationship where one system board can contain multiple I/O devices.

CIM Solaris_SystemBoardHasControllers Aggregation Properties

TABLE 4-20 CIM Solaris_SystemBoardHasControllers Aggregation Properties

Name	Data Type	Description
`GroupComponent`	REF Solaris_CHSystemBoard	References the parent in the relationship.
`PartComponent`	REF Solaris_CHController	References a child in the relationship.

CIM Process Indication Classes

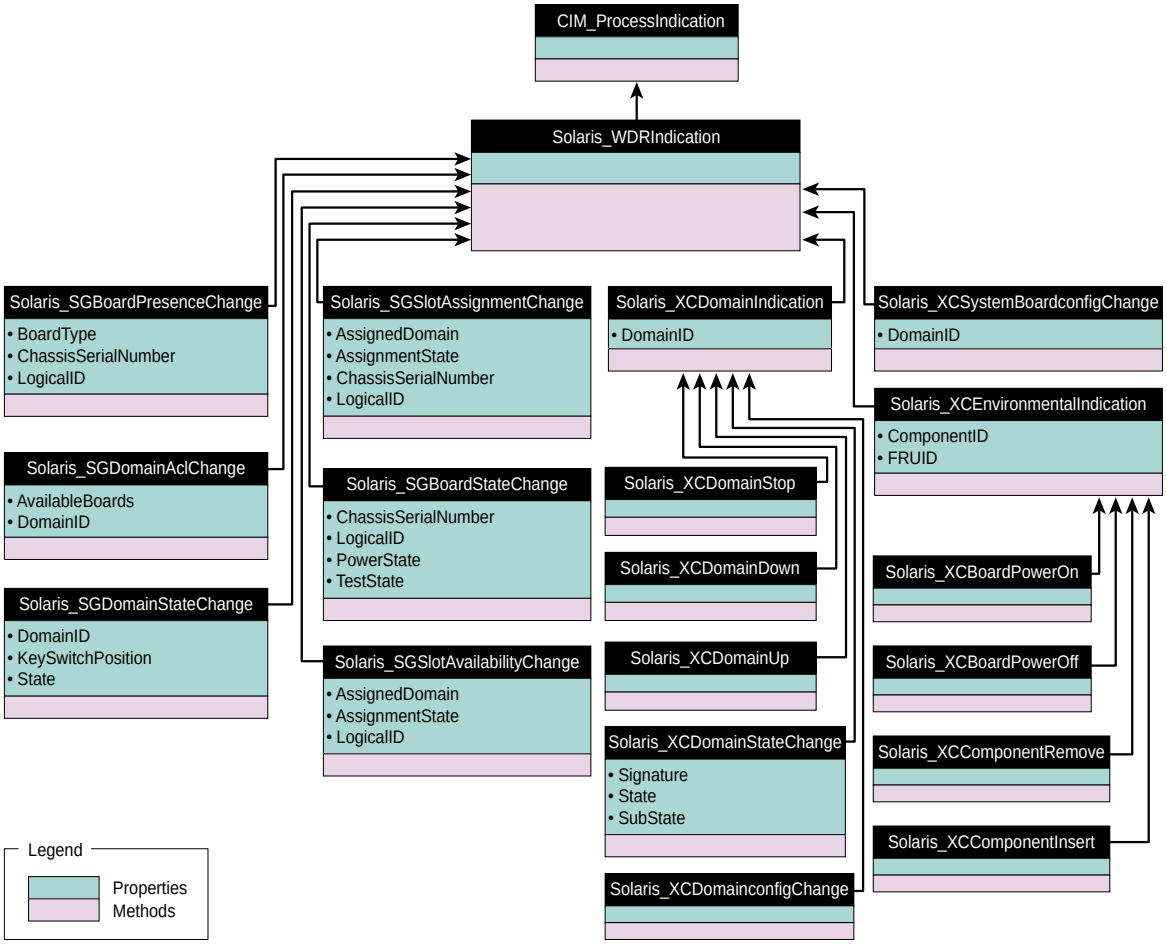
CIM process indications are subclasses of the `CIM_Processindication` class. They are used by WDR to forward notifications of events on Sun Fire 15K, 12K, 6800, 4810, 4800, and 3800 systems to client applications. Process indications are discussed fully in Chapter 3, “Using Process Indications.”

Process indications on Sun Fire 6800, 4810, 4800, and 3800 systems are derived from selected SNMP traps that are received from the System Controller (SC).

Process indications on Sun Fire 15K and 12K systems are derived from selected events that are generated by the system event facility, `sysevent`, on the Sun Fire 15K and Sun Fire 12K SC.

Note – Process indication classes whose names contain “XC” are used with Sun Fire™ 15K and 12K systems. Classes whose names contain “SG” are used with Sun Fire 6800, 4810, 4800, and 3800 systems.

The WDR Indication Class Hierarchy Diagram



Solaris_WDRIndication Class

The `Solaris_WDRIndication` class is an abstract class from which all process indication classes are derived on all Sun Fire systems. The `Solaris_WDRIndication` class adds no properties to its base class.

Solaris_SGBoardPresenceChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies a client that a CPU or an I/O board has become present or absent from a slot.

Direct Known Subclasses

None

Solaris_SGBoardPresenceChange Properties

TABLE 4-21 Solaris_SGBoardPresenceChange Properties

Name	Data Type	Description
`LogicalID`	string	The logical name of the slot. On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.
`ChassisSerialNumber`	string	The serial number of the chassis, which is an 8-digit hexadecimal string, such as `10483D99`.
`BoardType`	uint32	The type of board that occupies the slot is it is not empty. Possible values: `Unknown`, `Empty`, `CPU`, `IO`, `CPUWIB`, `IOWIB`, `SC`, `L2`, `Fan`, `Power Supply`, or `Logic Analyzer`. Currently, only boards of type `CPU` and `IO` are reported.

Solaris_SGDomainACLChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies the client that the Available Component List (ACL) has changed.

Direct Known Subclasses

None

Solaris_SGDomainACLChange Properties

TABLE 4-22 Solaris_SGDomainACLChange Properties

Name	Data Type	Description
`DomainID`	uint32	The domain to which the board was assigned, or from which it was unassigned. Possible values: `A`, `B`, `C`, or `D`.
`AvailableBoards[]`	boolean	The list of slots that are available to the domain that is identified by the `DomainID` property. Possible values: `SB0`, `SB1`, `SB2`, `SB3`, `SB4`, `SB5`, `IB6`, `IB7`, `IB8`, and `IB9`.

Solaris_SGDomainStateChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies the client that a domain goes up or down; that a domain self-test fails; or that the keyswitch state of a domain has changed.

Direct Known Subclasses

None

Solaris_SGDomainStateChange Properties

TABLE 4-23 Solaris_SGDomainStateChange Properties

Name	Data Type	Description
DomainID	uint32	The domain whose state has changed. Possible values: A, B, C, or D.
KeyswitchPosition	uint32	Identifies the keyswitch position of the virtual keyswitch. Possible values: Unknown, Off, Standby, On, Diag, Secure, Off To Standby, Off To On, Off To Diag, Off To Secure, Standby To Off, Active To Off, Active To Standby, Reboot To On, Reboot To Diag, and Reboot To Secure.
State	uint32	The current state of the domain. Possible values: Unknown, Running Post, Standby, Active, Powered Off, Domain Idle, Running OBP, Booting, Running Solaris, Halted, Reset, Panic, Debugger, or Hang Detected.

Solaris_SGSlotAssignmentChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies the client that a slot has been assigned to, or unassigned from, a domain.

Direct Known Subclasses

None

Solaris_SGSlotAssignmentChange Properties

TABLE 4-24 Solaris_SGSlotAssignmentChange Properties

Name	Data Type	Description
`LogicalID`	string	The logical name of the slot. On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.
`ChassisSerialNumber`	string	The serial number of the chassis, which is an 8-digit hexadecimal string such as 10483D99.
`AssignedDomain`	sint32	The domain to which the slot is assigned, if it is assigned. Possible values: `A`, `B`, `C`, or `D`, or `None`.
`AssignmentState`	uint32	The current assignment state of the slot. Possible values: `Unknown`, `Free`, `Assigned`, or `Active`.

Solaris_SGBoardStateChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies the client that a board self-test has completed, or that a board was powered-on or powered-off.

Direct Known Subclasses

None

Solaris_SGBoardStateChange Properties

TABLE 4-25 Solaris_SGBoardStateChange Properties

Name	Data Type	Description
`LogicalID`	string	The logical name of the slot. On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.
`ChassisSerialNumber`	string	The serial number of the chassis, which is an 8-digit hexadecimal string such as 10483D99.
`PowerState`	uint32	The power status of the board. Possible values: `Unknown`, `On`, `Off`, or `Failed`.
`TestState`	uint32	The test status of the board. Possible values: Unknown, Not Tested, Passed, Failed, Under Test, Start Test, Degraded, or Unusable.

Solaris_SGSlotAvailabilityChange Indication

This process indication, which is used on Sun Fire 6800, 4810, 4800, and 3800 systems, notifies the client that the slot's availability has changed.

Direct Known Subclasses

None

Solaris_SGSlotAvailabilityChange Properties

TABLE 4-26 Solaris_SGSlotAvailabilityChange Properties

Name	Data Type	Description
`LogicalID`	string	The logical name of the slot. On a Sun Fire 6800, 4810, 4800, or 3800 system there can be up to 6 system boards, whose slots are represented as SB0, SB1, ... SB5; and up to 4 I/O boards, whose slots are represented as IB6, IB7, IB8, and IB9.
`AssignedDomain`	sint32	The domain to which the slot was assigned, and from which it is now unassigned; or the domain to which the slot has been assigned. Possible values: `A`, `B`, `C`, or `D`.
`AssignmentState`	uint32	The current assignment state of the slot. Possible values: `Unknown`, `Free`, `Assigned`, or `Active`.

Solaris_XCSystemBoardConfigChange Indication

This process indication, which is used only on Sun Fire 15K and 12K systems, notifies the client that one or more Sun Fire 15K/12K domain configuration properties has changed for a specific domain.

Direct Known Subclasses

None

Solaris_XCSystemBoardConfigChange Properties

TABLE 4-27 Solaris_XCSystemBoardConfigChange Properties

Name	Data Type	Description
`LogicalID`	string	Identifies the system board whose configuration data has changed.

Solaris_XCEnvironmentalIndication Indication

An abstract class that serves as a common ancestor to all environmental indications on Sun Fire 15K and 12K systems.

Direct Known Subclasses

None

Solaris_XCEnvironmentalIndication Properties

The `Solaris_XCEnvironmentalIndication` class adds the following properties to its base class:

TABLE 4-28 Solaris_XCEnvironmentalIndication Properties

Name	Data Type	Description
`ComponentID`	string	The component that is experiencing the environmental event
`FRUID`	uint32	If the component is a system board, contains the corresponding Field Replaceable Unit identifier; otherwise NULL.

Solaris_XCComponentRemove Indication

Derived from the `Solaris_XCEnvironmentalIndication` abstract class, this class notifies a client that a specific hot-pluggable component has been removed from its slot on a Sun Fire 15K or 12K system.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCComponentInsert Indication

Derived from the `Solaris_XCEnvironmentalIndication` abstract class, this class notifies a client that a specific hot-pluggable component has been inserted into its slot on a Sun Fire 15K or 12K system.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCBoardPowerOn Indication

Derived from the `Solaris_XCEnvironmentalIndication` abstract class, this class notifies a client that a system board has been powered-on in a Sun Fire 15K or 12K system.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCBoardPowerOff Indication

Derived from the `Solaris_XCEnvironmentalIndication` abstract class, this class notifies a client that a system board has been powered-off in a Sun Fire 15K or 12K system.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCDomainIndication Indication

Derived from the `Solaris_XCEnvironmentalIndication` abstract class, this abstract class that serves as a common ancestor to all domain indications on Sun Fire 15K and 12K systems.

Direct Known Subclasses

None

Solaris_XCDomainIndication Properties

The `Solaris_XCDomainIndication` class adds the following property to its base class:

TABLE 4-29 Solaris_XCDomainIndication Properties

Name	Data Type	Description
`DomainID`	uint32	Identifies the domain that is experiencing the event.

Solaris_XCDomainConfigChange Indication

Derived from the `Solaris_XCDomainIndication` abstract class, this class notifies a client that one or more configuration properties have been changed in a specific domain on a Sun Fire 15K or 12K system.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCDomainUp Indication

Derived from the `Solaris_XCDomainIndication` abstract class, this class notifies a client that a specific domain has gone up on a Sun Fire 15K or 12K system. A domain goes up when the keyswitch is set to On; or after the domain monitoring daemon, DSMD, is re-started and finds that the IOSRAM that is assigned to the domain is accessible.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCDomainDown Indication

Derived from the `Solaris_XCDomainIndication` abstract class, this class notifies a client that a specific domain has gone down on a Sun Fire 15K or 12K system. A domain goes down when the keyswitch is set to Off or Standby.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCDomainStop Indication

Derived from the `Solaris_XCDomainIndication` abstract class, this class notifies a client that a specific domain on a Sun Fire 15K or 12K system has begun a hardware state dump. A hardware state dump occurs when a non-recoverable hardware failure causes the domain to write its state information to a dump file.

This class adds no properties to its base class and has no direct known subclasses.

Solaris_XCDomainStateChange Indication

Derived from the `Solaris_XCDomainIndication` abstract class, this indication notifies the client that the state of a specific domain on a Sun Fire 15K or 12K system has changed.

Direct Known Subclasses

None

Solaris_XCDomainStateChange Properties

The `Solaris_XCDomainStateChange` class adds the following property to its base class:

TABLE 4-30 Solaris_XCDomainStateChange Properties

Name	Data Type	Description
`Signature`	uint32	The `Signature`, `State`, and `SubState` properties combine to describe the current state of the domain.
`State`	uint32	The `Signature`, `State`, and `SubState` properties combine to describe the current state of the domain.
`SubState`	uint32	The `Signature`, `State`, and `SubState` properties combine to describe the current state of the domain.

CHAPTER 5

Programming Techniques in WDR

This chapter provides code examples that illustrate techniques for performing tasks using WDR. However, these examples are not intended for use in production WDR applications.

The code examples demonstrate how you work with providers:

- EventProvider
- InstanceProvider
- AssociatorProvider
- MethodProvider

Caching System State Information

An important consideration when developing client applications for WDR is that there are two fundamentally different possible approaches to ensure that the client has a knowledge of the current state of the domains, attachment points and slots of the managed platform: polling and using cache.

The client can periodically poll for the status of domains, attachment points and slots, by enumerating the instances of the corresponding WDR classes. This approach is not recommended, since the time taken to execute an operation using WDR is dependent on the system state and workload, and can be variable. This will adversely affect the performance of both the System Controller (SC) and the client application.

A better approach is for the client to maintain a current cache of the domain, attachment point and slot status, and use the WDR Process Indications to indicate when updates to the client's cache of status information are necessary. See the section "CIM Process Indication Classes" on page 82 for more information.

Working with an EventProvider

The following example demonstrates how to create an EventProvider.

Indication Reader

The following code shows how to subscribe to, and to read, WDR event indications:

```
/* Standard java packages */
import java.io.*;
/* Solaris WBEM packages */
import com.sun.wbem.cim.*;
import com.sun.wbem.client.*;
import com.sun.wbem.security.*;

public class IndicationReader
{
    public static void main(String args[]) throws CIMException
        {
            if (args.length != 3) {
                System.out.println("Usage: java IndicationReader " +
                    "<hostname> <username> <password>");
            System.exit(1);
        }
        String hostName = args[0];
        UserPrincipal userName = new UserPrincipal(args[1]);
        PasswordCredential passWord = new PasswordCredential(args[2]);
        CIMNameSpace nameSpace = new CIMNameSpace();
        nameSpace.setHost(hostName);
        // Read all WDR Indications.
        final String filter = "SELECT * FROM Solaris_WDRIndication";
        IndicationSubscription subscription = null;
        try
        {
            // creates a CIMClient adding CIMListener to it.
```

```
            CIMClient cc = new CIMClient(nameSpace, userName,
                passWord);
            cc.addCIMListener(new EventListener());
            // subscribes to WDR Indications and waits
            subscription = new IndicationSubscription(cc, filter);
            System.out.println("Waiting for Indications...");
            waitForQuit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            if ( subscription != null ) {
                subscription.remove();
            }
        }
    System.exit(0);
    }
    /*
    * Exit when user types 'quit'
    */
    private static void waitForQuit() throws IOException
    {
        BufferedReader stdin =
        new BufferedReader( new InputStreamReader(System.in));
        String line = null;
        do {
            System.out.println("Type 'quit' followed by <CR> to exit");
            System.out.print("IR> ");
            line = stdin.readLine();
        } while ( ! line.startsWith("quit") );
    }
}
```

Event Listener

The following code inplements the CIMListener interface so that it can listen for CIM events. To register for indications of CIM events, the client must add an instance of CIMListener.

```
/* WBEM libraries */
import com.sun.wbem.client.*;

public class EventListener implements CIMListener
{
    public EventListener()
    {
    }
    /**
    * Prints indication of an event when the indication is available
    * for delivery.
    */
    public void indicationOccured(CIMEvent e)
    {
        System.out.println("Received " + e.getIndication());
    }
}
```

Indication Subscription

The IndicationSubscription class enables clients to subscribe to be notified of CIM events. It binds an event filter to an event handler.

```
/* Standard Java packages */
import java.util.*;

/* Standard WBEM packages */
import com.sun.wbem.cim.*;
import com.sun.wbem.client.*;
import com.sun.wbem.security.UserPrincipal;
import com.sun.wbem.security.PasswordCredential;
```

```
public class IndicationSubscription
{
    static protected int m_FilterCnt = 0;

    protected CIMClient m_Client;
    protected CIMObjectPath m_Filter;
    protected CIMObjectPath m_Handler;
    protected CIMObjectPath m_Subscription;

    final String subscriptionClassName =
       "CIM_IndicationSubscription";
    final String filterClassName = "CIM_IndicationFilter";
    final String deliveryClassName = "Solaris_RMIDelivery";
    /**
    * Force construction through another constructor that is public.
      */
    protected IndicationSubscription() {
       m_Client = null;
       m_Filter = null;
       m_Handler = null;
       m_Subscription = null;
    }

    /**
    * Construct an IndicationSubscription that subscribed for
       Indications as expressed by the specified filterExp.  Three
    * CIM objects are created in the CIM repository as a
    * side-effect of calling this method, a CIM_IndicationFilter,
    * a CIM_IndicationHandler, and a CIM_IndicationSubscription.
    * These can be removed by calling the remove method.
    *
    * @param cc            a CIMClient instance
    * @param filterExp     The query string on which to filter
                             Indications
    * @exception CIMException
    */
    public IndicationSubscription(CIMClient cc, String filterExp)
```

```
throws CIMException
{
   m_Client = cc;
   m_Filter = createFilter(filterExp);
   m_Handler = createHandler();
   m_Subscription = createSubscription();
}
/**
* Removes the otherwise persistant filter, handler and
* subscription CIM objects from the CIM repository.
* @exception CIMException  if an attempt is made to delete a
* non-existent CIM object.
*/
public void remove() throws CIMException {
   if ( m_Subscription != null ) {
      m_Subscription.setNameSpace("");
      m_Client.deleteInstance(m_Subscription);
      m_Subscription = null;
   }
   if ( m_Handler != null ) {
      m_Handler.setNameSpace("");
      m_Client.deleteInstance(m_Handler);
      m_Handler = null;
   }
   if ( m_Filter != null ) {
      m_Filter.setNameSpace("");
      m_Client.deleteInstance(m_Filter);
      m_Filter = null;
   }
}

/**
* Create an IndicationFilter of the specified name and with the
* specified filterExp as the query string.  Register the filter
* by creating its instance in the repository.  Only one filter
* may exist per IndicationSubscription object.
*
```

```
* @param filterExp The query string on which to filter
* Indications
* @return CIMObjectPath of the filter.
* @exception CIMException
*/
protected CIMObjectPath createFilter(String filterExp) throws
   CIMException
{
    CIMClass filterClass =
        m_Client.getClass(new CIMObjectPath(filterClassName),
        false, true, true, null);

    CIMInstance ci = filterClass.newInstance();

    ci.setProperty("Name", new CIMValue(generateFilterName()));
    ci.setProperty("Query", new CIMValue(filterExp));
    ci.setProperty("QueryLanguage", new CIMValue("WQL"));

    CIMObjectPath op = m_Client.createInstance(new
       CIMObjectPath(), ci);
    return ( op );
}
/**
* Generate a unique filter name for this Java VM.
*
* @return Name of the filter.
*/
protected String generateFilterName()
{
    String filterName = "WDRFilter"+ m_FilterCnt;
    m_FilterCnt = (m_FilterCnt + 1) % Integer.MAX_VALUE;
    return ( filterName );
}

/**
* Create an indication handler.
* Register the handler by creating its instance in the repository.
```

```
*
* @return CIMObjectPath of the handler.
*/
protected CIMObjectPath createHandler() throws CIMException
{
   CIMClass deliveryClass =
   m_Client.getClass(new CIMObjectPath(deliveryClassName),
                           false, true, true, null);
   CIMInstance ci = deliveryClass.newInstance();

   CIMObjectPath op = m_Client.createInstance(new
      CIMObjectPath(), ci);
   return ( op );
}
/**
* Create an indication subscription that binds filter to handler.
* Register the subscription by creating its instance in the
     repository.
*
* @return CIMObjectPath of subscription.
*/
protected CIMObjectPath createSubscription() throws CIMException
{
   final String subscriptionClassName =
      "CIM_IndicationSubscription";
   CIMClass subscriptionClass =
   m_Client.getClass(new CIMObjectPath(subscriptionClassName),
      false, true, false, null);

   CIMInstance ci = subscriptionClass.newInstance();
   ci.setProperty("Filter", new CIMValue(m_Filter));
   ci.setProperty("Handler", new CIMValue(m_Handler));

   m_Client.createInstance(new CIMObjectPath(), ci);

   // we are looking for the subscription's reference because
```

```
        // createInstance() returns a null reference for the
              subscription.
        CIMObjectPath cop =
           new CIMObjectPath(subscriptionClassName,
                ci.getKeyValuePairs());
        return ( cop );
    }
}
```

Working with an InstanceProvider

The following code samples assume that a CIMClient object called `m_Client` has already been created and is available for use.

The first code sample gets all instances of the `Solaris_XCDomain` class using the `enumerateInstanceNames` and `getInstance` methods:

```
// gets path to all instances
CIMObjectPath cop = new CIMObjectPath("Solaris_XCDomain");
Enumeration e = m_Client.enumerateInstanceNames(cop);

// gets instances from the instances' paths
while ( e.hasMoreElements() ) {
     cop = (CIMObjectPath) e.nextElement();
     CIMInstance ci = m_Client.getInstance(cop, true, false, false,
       null);
     System.out.println(ci.toString());
}
```

The second code sample demonstrates how to invoke the `enumerateInstances` method:

```
CIMObjectPath cop = new CIMObjectPath("Solaris_XCDomain");
Enumeration e = m_Client.enumerateInstances(cop, true, false, false,
       null);

while ( e.hasMoreElements() ) {
     CIMInstance ci  = (CIMInstance) e.nextElement();
     System.out.println(ci.toString());
```

```
}
```

Working with an AssociatorProvider

The following code samples assume that a CIMClient object called m_Client has been created and is available for use.

The first example gets each instance of the Solaris_CHCPU class that is associated with an instance of the `Solaris_CHSystemBoard` class via the `Solaris_SystemBoardHasProcessors` association:

```
// sbCOP is a CIMObjectPath of a system board.
String assocClass = "Solaris_SystemBoardHasProcessor";
String resultClass = "Solaris_CHCPU";
String role = "SystemBoard";
String resultRole = "Processor";
boolean includeQualifiers = true;
boolean includeClassOrigin = true;
String[] cpuProperty = null;


Enumeration e = m_Client.associators(sbCOP, assocClass, resultClass,
      role, resultRole, includeQualifiers, includeClassOrigin,
      cpuProperty);
while ( e.hasMoreElements() ) {
   CIMInstance ci = (CIMInstance) e.nextElement();
   System.out.println(ci.toString());
}
```

The second example enumerates association objects that refer to an instance of the `SolarisCHSystemBoard` class and to instances of the `Solaris_CHCPU` class:

```
// cop is CIMObjectPath of the Solaris_CHSystemBoard instance
String resultClass = "Solaris_SystemBoardHasProcessors"
String role = "SystemBoard";
String includeQualifiers = true;
String includeClassOrigin = true;
String[] propertyList = "Processor";
```

```
Enumeration e = m_Client.references(cop, resultClass, role,
includeQualifiers, includeClassOrigin, propertyList);
while ( e.hasMoreElements() ) {
   CIMInstance assoc = (CIMInstance) e.nextElement();
   System.out.println(assoc.toString());
}
```

Working with a MethodProvider

The following code samples assume that a `CIMClient` object called `m_Client` has been created and is available for use.

The first example configures a single processor and prints out to the standard output any error messages that may occur during the configuration process:

```
// cop is CIMObjectPath of the processor
String method = "configure";
Vector inParams = new Vector(4);
Vector outParams = new Vector(2);

inParams.add(CIMValue.FALSE);                 /* force */
inParams.add(new CIMValue(new String("")));   /* hwOptions */
inParams.add(new CIMValue(new Integer(3)));   /* 3 retries */
inParams.add(new CIMValue(new Integer(5)));   /* 5s delay */

CIMValue returnVal = m_Client.invokeMethod(cop, method, inParams,
      outParams);
int status = ((Integer)(returnVal.getValue())).intValue();
if ( status != 0 && outParams.size() != 0 ) {
    Object obj = ((CIMValue)(outParams.elementAt(0))).getValue();
    String error = (String) obj;
    if ( error != null ) {
       System.out.println(error);
    }
}
```

The second code sample assigns a system board to a domain and prints to the standard output any error messages that may occur during the assignment process:

```
// cop is the CIMObjectPath of a system board
String method = "Assign";
Vector inParams = new Vector(1);
Vector outParams = new Vector(2);
inParams.add(new CIMValue(new Integer(domainID))); /* domainID
CIMValue returnVal = m_Client.invokeMethod(cop, method, inParams,
       outParams);
int status = ((Integer)(returnVal.getValue())).intValue();
if ( status != 0 && outParams.size() != 0 ) {
   Object obj = ((CIMValue)(outParams.elementAt(0))).getValue();
   String error = (String) obj;

   if ( error != null ) {
      System.out.println(error);
   }
}
```

Index

D

E

F

H

U

W