Sun™ SNMP Management Agent Administration Guide for Sun Supported Servers

Version 1.5.3

Sun Microsystems, Inc.
www.sun.com

Part No. 819-7978-13
January 2008, Revision A

Submit comments about this document at: http://www.sun.com/hwdocs/feedback

Please
Recycle

Contents

Figures

Tables

Preface

This guide describes the Sun SNMP Management Agent (SMA), which supports management of hardware using the Simple Network Management Protocol. The SNMP Management Agent provides monitoring of inventory, configuration, service indicators, and environmental and fault reporting.

It is intended to be read by experienced enterprise administrators and professional developers.

The guide:

- Explains how to install and configure the software.
- Introduces the SNMP Management Agent and describes its functionality.

Refer to the version of *Sun SNMP Management Agent Release Notes* that corresponds to the version of the software you are using for specific information about your software release.

How This Document Is Organized

The guide contains the following chapters:

Chapter 1 provides an overview of the software.

Chapter 2 describes how to install the management software.

Chapter 3 provides information about the user-configurable files.

Chapter 4 provides help in troubleshooting your software.

Chapter 5 explains how to uninstall the software.

Chapter 6 describes static platform-specific information, including `sunPlat` class definitions and logical and log table population.

Chapter 7 provides a brief introduction to the essential features of the Simple Network Management Protocol.

Chapter 8 provides an overview of how SNMP models the hardware platform.

Chapter 9 describes how the managed objects and their relationships are presented by the SNMP interface.

Chapter 10 describes the sunPlat physical class hierarchy and how the managed physical object classes defined in the sunPlat model are represented by the `SUN-PLATFORM-MIB`.

Chapter 11 describes the notifications classes and attributes, as defined in the `SUN-PLATFORM-MIB`.

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

  `http://docs.sun.com`

Shell Prompts

Shell	Prompt
C shell	*machine-name*%
C shell superuser	*machine-name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Typographic Conventions

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. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type rm *filename*.

* The settings on your browser might differ from these settings.

Related Documentation

Documents specifically related to your Solaris operating system or your platform can be found online at:

http://docs.sun.com/

Documentation, Support, and Training

Sun Function	URL
Documentation	http://www.sun.com/documentation/
Support	http://www.sun.com/support/
Training	http://www.sun.com/training/

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

http://www.sun.com/hwdocs/feedback

Please include the title and part number of your document with your feedback: *Sun SNMP Management Agent Administration Guide for Sun Supported Servers*, part number 819-7978-13

CHAPTER 1

Overview

The Sun SNMP Management Agent enables access to system inventory and monitoring and provides support for alarms, using the industry standard management protocol Simple Network Management Protocol (SNMP). The agent supports SNMPv1, SNMPv2c and SNMPv3 to enable interoperability with all common management applications. The provision of SNMPv3 enables management accesses to be fully authenticated and secured.

The agent provides a management model based on the standard `ENTITY-MIB`, and is augmented by extensions that provide further information dependent on the component being represented. These extensions are based on the generic network information model (NIM) presented in ITU-T M.3100 with further extensions taken from attributes defined by the common information model (CIM) v2.5 schema. These Management Information Bases (MIBs) are supported on other platforms, enabling common management solutions to be developed.

The agent provided is intended to augment the management information, such as MIB-II, already presented by the standard Solaris SNMP agent, `snmpdx`. Both agents run independently of each other.

For a list of supported platforms in a specific software release, refer to the version of the *Sun SNMP Management Agent Release Notes* that corresponds to the version of the software you are using.

CHAPTER 2

Installation

This chapter describes how to install the management software.

The chapter contains the following sections:

System Requirements

Before installing SNMP Management Agent, ensure that your system complies with the prerequisites and dependencies discussed in this section.

Solaris OS Requirements

Refer to the system manuals for your platform for Solaris operating system (OS) requirements. See "Related Documentation" on page xiv of the Preface.

Disk Space Requirements

At least 10 megabytes must be available on the server.

Installation and Additional Required Packages

The installation packages in table 2-1 and the additional required packages listed in table 2-2 must be installed to enable support of the SNMP Management Agent. For the location of these pages, refer to "Location of Installation and Additional Required Packages" in the *Sun SNMP Management Agent Release Notes, Version 1.5.3*.

Installation Packages

TABLE 2-1 lists all packages contained in the tar archive bundle. However, the Sun Fire V1280, E2900, and Netra 1280 and 1290 servers require only the `SUNWmasf` and `SUNWmasfr` packages.

TABLE 2-1 SNMP Management Agent Software Installation Package Descriptions

Package	Package Name	Function
`SUNWespdl`	Common Config Reader PCPDAQ Library	Framework for providing configuration information for Sun Blade T6300, Sun Fire T1000/T2000, and Netra T2000
`SUNWescdl`	Common Config Reader DAQ Library	Framework for providing configuration information
`SUNWescpl`	Common Config Reader Sun Fire V125/V210/V215/V240/V245 and Netra 240/210 platform support	Configuration reader platform support for the Sun Fire V125,V210,V215,V240,V245 and Netra 240/210
`SUNWescfl`	Common Config Reader Sun Fire V250 platform support	Configuration reader platform support for the Sun Fire V250
`SUNWescnl`	Common Config Reader Sun Fire V440/445 platform support	Configuration reader platform support for the Sun Fire V440/V445
`SUNWeschl`	Common Config Reader Netra 440/445 platform support	Configuration reader platform support for the Netra 440
`SUNWeserl`	Common Config Reader Sun Fire T1000 platform support	Configuration reader platform support for Sun Fire T1000

TABLE 2-1 SNMP Management Agent Software Installation Package Descriptions *(Continued)*

Package	Package Name	Function
SUNWesonl	Common Config Reader T2000/Netra T2000 /Sun Blade T6300 and SPARC Enterprise T5120/T5120platform support	Configuration reader platform support for Sun Fire T2000/Netra T2000/Sun Blade T6300 and SPARC Enterprise T5120/T5220
SUNWmasf	Sun SNMP Management Agent for Sun Fire, Netra, Sun Blade and Sun SPARC Enterprise systems	SNMP Agent common components
SUNWmasfr	Sun SNMP Management agent for Sun Fire, Netra, Sun Blade and Sun SPARC Enterprise systems	Startup and configuration scripts for the SNMP agent

Additional Required Packages

The packages listed in TABLE 2-2 must be installed to enable support of the SNMP Management Agent.

TABLE 2-2 SNMP Management Agent Additional Required Software Packages

Package	Package Name
SUNWfruid	FRU ID Utility and Library (Usr)
SUNWfruip	FRU ID Platform Modules (Usr)
SUNWmfrun	Motif RunTime Kit
SUNWpiclh	PICL Header Files (Usr)
SUNWpiclr	PICL Framework (Root)
SUNWpiclu	PICL Libraries, and Plugin Modules (Usr)

To upgrade the software, you must remove the existing software before reinstalling the new version (see Chapter 5).

Effects on System Files

A new startup file is created in `/etc/init.d`, as shown in TABLE 2-3, with links to `/etc/rc<n>.d`..

TABLE 2-3 Startup Script

Component	Startup Script	Package Name	Package Description
Agent	`masfd`	`SUNWmasfr`	Configuration and startup script for SNMP agent

After installation, the following MIBs, supported by the agent, are located in the directory `/opt/SUNWmasf/lib/mibs`:

TABLE 2-4 MIB Files

MIB	Function
`ENTITY-MIB.txt`	Describes physical and logical entities
`RFC1155-SMI.txt`	Defines additional object types used by other MIBS
`RFC1213-MIB.txt`	Models network interfaces (note, this agent only supports system part)
`SUN-PLATFORM-MIB.txt`	Extends the Entity MIB to provide additional information about hardware components

Installing the SNMP Software

This section describes the procedure for installing the monitoring software.

Before installing the software, ensure that:

- You have the requisite level of the Solaris OS installed on both the domain or target and the platform agent server (see "Solaris OS Requirements" on page 3).
- You have installed all the necessary patches (refer to "Patches" in the *Sun SNMP Management Agent Release Notes, Version 1.5.3*) and any additional essential packages (TABLE 2-2) not supplied as part of the SNMP software,.

When you are certain that your system meets all these requirements, you can proceed to install the SNMP software.

▼ To Install the SNMP Agent

1. **Unzip the *SNMP-zip* file by typing:**

```
$ zcat unzip SNMP-zip
```

2. **Ensure you are the superuser before proceeding with installation.**

3. **Change to the directory where the zip file was extracted (at the root of the packages provided by the zip file).**

   If you wish to start installation from a different directory, when using `pkgadd` you must specify the path of the root of the directory hierarchy where the files where extracted using the `-d` option (rather than `-d .` as shown in step 4).

Caution – If you have installed Sun™ Management Center on your monitored platform, skip to Step 5.

---

4. **Install the support for platform instrumentation:**

   For the Sun Fire V1280, E2900 and Netra 1280 and 1290 servers, skip to Step 5.

   - To install the Sun Fire V125/V210/V215/V240/V245 and Netra 210/240 instrumentation, type:

```
# pkgadd -d . SUNWescdl SUNWescpl
```

   - To install the Sun Fire V250 instrumentation, type:

```
# pkgadd -d . SUNWescdl SUNWescfl
```

   - To install the Sun Fire V440 and V445 instrumentation, type:

```
# pkgadd -d . SUNWescdl SUNWeschl
```

   - To install the Netra 440 instrumentation, type:

```
# pkgadd -d . SUNWescdl SUNWescnl
```

   - To install the Sun Fire T1000 instrumentation, type:

```
# pkgadd -d . SUNWespdl SUNWescdl SUNWeserl
```

   - To install the Sun Fire T2000, Sun Blade T6300 and Sun SPARC Enterprise T5120/T5220 instrumentation, type:

```
# pkgadd -d . SUNWespdl SUNWescdl SUNWesonl
```

- To install the Netra T2000 instrumentation, type:

```
# pkgadd -d . SUNWespdl SUNWescdl SUNWesonl
```

5. **Install the SNMP agent by typing:**

```
# pkgadd -d . SUNWmasf SUNWmasfr
```

▼ To Configure the SNMP Agent

After installation, you must configure the agent before you attempt to start it. A full description of the configuration options for the SNMP agent is provided in Chapter 3.

Before using the agent, you must assign it a network port to use. In deciding which network port to use, consider:

- Whether other SNMP agents may be installed on the system
- Where management application(s) may attempt to discover SNMP agents

---

Note – Use a port other than 161 for SNMP access to the agent provided by this product to ensure that this port is available for use by other agents, such as `snmpdx` and Sun Management Center. The conventional port for SNMP access is port 161 for data. By default, the Solaris `snmpdx` daemon uses this port.

---

1. **Configure the SNMP agent using the `/etc/opt/SUNWmasf/conf/snmpd.conf` file.**
2. **To set the port number, add a line to the file in the form:**

```
agentaddress     9161
```

This enables the agent to communicate on the port 9161. Make sure that the port you select is not currently in use by another application. If you are using SNMPv1 or SNMPv2c, no further configuration is required unless you wish to use traps, in which case you must specify the trap destinations. See Chapter 3 for more information.

▼ To Start the SNMP Agent

1. **Type:**

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

2. **Confirm that the agent is correctly running by typing:**

```
# ps -ef | fgrep snmpd
```

You should see a line of the form:

```
root 29394  1 0 Feb 18 ?  4:11 /opt/SUNWmasf/sbin/snmpd
```

If you do not see this output, review the `/var/adm/messages` log file to locate any error messages from the agent. For more information on troubleshooting, see Chapter 4.

On subsequent reboots the agent starts automatically without any user intervention.

CHAPTER 3

Configuration Files

This chapter describes how to configure the SNMP agent.

The chapter contains the following sections:

Overview

You can configure the SNMP agent with the `/etc/opt/SUNWmasf/conf/snmpd.conf` file. This file defines the ports where the agent can be accessed, the access controls that it is to apply to management information, and destinations for traps. By default, only the port number requires configuring after installation if:

- The agent is only to be used for SNMPv1 and/or SNMPv2c accesses
- The default community is acceptable
- No trap destinations are required

The default configuration used by the agent enables read-only access for the community *public*. If this is not required or it is inappropriate for a particular environment, configure access control as described below.

If there are systems that are to receive traps, provide a list of such hosts to the agent as described in "Setting Up Trap and/or Inform Destinations" on page 13.

If SNMPv3 access is required, further configure SNMPv3 and SNMPv3 users and views as described in "SNMPv3 Configuration" on page 20.

If the management applications using the agent are likely to require information in the MIB-II system branch, refer to "Setting System Information" on page 21.

You can add comments to the configuration file by making the pound sign (#) the first character on the comment line.

Setting Up the Port Number

Configure the port number using the `agentaddress` keyword:

```
agentaddress [<transport-specifier>:]<transport-address> [,...]
```

This makes the agent listen on the specified comma-separated list of listening addresses. At least one port must be specified for the agent to function. The transport-specifier should always be set to `udp`. The transport address is of the form:

```
hostname[:<port>]
```

or

```
<IPv4-address>[:<port>]
```

For example, specifying agent address as:

```
agentaddress 8161,localaddress:9161
```

makes the agent listen on UDP port 8161 on all IPv4 interfaces and UDP port 9161 only on the interface associated with the localhost address.

The agent *must* have a port number configured before it will start. No default value is provided for this port number and its value must be determined by the administrator on the basis of other agents that may be in use and the configuration of the management clients.

Note – By using ports other than 161 and 162 for SNMP agent access, you can ensure that these ports are available for use by other agents such as `snmpdx` or Sun Management Center, which use them by default.

Setting Up Trap and/or Inform Destinations

Syntax

```
trapcommunity <string>
```

This defines the default community string to be used when sending traps.

Note – You must use the `trapcommunity` command before any of the following three commands, which use this community string.

If you do not set a community string, the default value `public` is used.

```
trapsink <host>[<community>[<port>]]

trap2sink <host>[<community>[<port>]]

informsink <host>[<community>[<port>]]
```

These commands define the hosts to receive traps (and/or inform notifications):

- Use `trapsink` to specify a host to be sent SNMPv1 traps.
- Use `trap2sink` to send SNMPv2 traps.
- Use `informsink` to send inform notifications.

You can use multiple `trapsink`, `trap2sink` and `informsink` lines to specify multiple destinations. You can also specify a host multiple times, in which case a trap is sent for each of the commands listed.

The daemon sends a cold start trap when it starts up. If enabled, it also sends traps on authentication failures. If <*community*> is not specified, the string from a preceding trap community directive is used. If no community has been specified, the value `public` is used. If <*port*> is not specified, the well-known SNMP trap port (162) is used.

CODE EXAMPLE 3-1 Using the `trapsink` and `trap2sink` Commands

```
....
agentaddress 8161,localhost:9161
#############################################################
# SECTION: Trap Destinations
#
# Here we defined who the agent will send traps to.

# trapsink: A SNMPv1 trap receiver
# arguments host [community] [portnum]
trapsink merlin public 32768

# trap2sink: A SNMPv2 trap receiver
# arguments host [community] [portnum]
trap2sink arthur
```

This results in SNMPv1 traps being sent to the system *merlin* on port 32768, and SNMPv2 traps being sent to *arthur* using the default port for traps of 162.

Configuring Access Control

The agent supports the view-based access control model (VACM) as defined in RFC 2575. It therefore recognizes the following keywords in the configuration file:

- `com2sec`
- `group`
- `access`
- `view`

It also recognizes some easier-to-use wrapper directives:

- `rocommunity`
- `rwcommunity`
- `rouser`
- `rwuser`

Syntax

rocommunity and rwcommmunity

```
rocommunity <community>[<source>] [<OID>]

rwcommunity <community>[<source>] [<OID>]
```

These create read-only and read-write communities that can be used to access the agent. They are a simple wrapper around the more complex and powerful com2sec, group, access, and view directive lines. They are also not as efficient because groups are not created and the tables can be larger as a consequence. It is therefore better not to use these if you have complex situations to set up, and to reserve their use to where your setup is simple.

- The <*community*> token specifies the community string for which access is to be granted.
- The format of the <*source*> is token and is described in the com2sec directive section.
- The <*OID*> token restricts access for that community to everything below the given OID.

You can apply only one rwcommunity or rocommunity directive for each *source/community* combination.

rouser and rwuser

```
rouser <username> [noauth|auth|priv] [<OID>]

rwuser <username> [noauth|auth|priv] [<OID>]
```

These configure access permissions for the specified user in the VACM configuration tables. It is more efficient (and powerful) to use the combined group, access, and view directives instead, but these wrapper directives are much simpler.

- <*username*> specifies the SNMPv3 security name to which these permissions apply.
- The minimum level of authentication and privacy for <*username*> is specified by the first token (which defaults to auth).
- The <*OID*> token restricts access for that user to everything below the given OID.

These directives have no effect unless the corresponding user has been created (see "SNMPv3 Configuration" on page 20).

You can apply only one `rwuser` or `rouser` directive for each user.

com2sec

```
com2sec <username> <source> <community>
```

This directive specifies the mapping from a *source/community* pair to a security name.

- <*username*> specifies the security name to which this source and community string are to be mapped.
- <*source*> can be a hostname, a subnet, or the word `default`. A subnet is specified as IP/mask (for example, `129.156.203.56/255.255.255.0`) or IP/bits (for example, `129.156.203.56/24`).
- `default` specifies that access is to be provided to all hosts with the specified community.

The first *source/community* combination that matches the incoming packet is selected.

group

```
group <groupname> <model> <username>
```

This directive defines the mapping from a given security model and security name to a particular group.

- <*groupname*> defines the name of the group to which this combination of model and security name belongs.
- <*model*> is the security model and can be any one of `v1`, `v2c`, or `usm`.
- <*username*> specifies the security name, which is defined elsewhere either in a `com2sec` or `createUser` directive.

access

```
access <groupname> <context> <model> <level> <match> <read> <write> <notify>
```

The access directive specifies the access permissions to be given to a particular group/security model combination.

- <*groupname*> specifies the name of the group to which the directive applies.
- <*model*> is the SNMP security model to which the access directive applies. It can take the values of any, v1, v2c, or usm.
- <*level*> specifies the minimum level of authentication and encryption of the incoming requests for which this directive applies. It can take the values noauth, auth, or priv.
- <*match*> specifies how <*context*> should be matched.
- against the context of the incoming pdu, and takes the values exact or prefix.
  - exact specifies that <*context*> must be matched exactly .
  - prefix specifies that the context must begin with <*context*>.
- <*read*>, <*write*>, and <*notify*> specify the view to be used for the corresponding access.

Note – The <*notify*> field is ignored for the purposes of access control. All access control for traps and informs is configured by means of the trapsink, trap2sink, and informsink directives (see "Setting Up Trap and/or Inform Destinations" on page 13).

When <*model*> is v1 or v2c, set <*level*> to noauth, and <*context*> to the empty string "".

When access for a particular group with a given security model is requested, the agent determines access in the following order:

1. Entries with an matching security model (as opposed to those matching against any).
2. If there is still more than one match, the most exact <*context*> match.
3. If there is still more than one match, the entry with the highest security level.

view

```
view <viewname> <type> <subtree> [<mask>]
```

This defines the named view.

- <*viewname*> defines the name of the view, which you can then use to reference the view in an `access` directive.
- <*type*> takes the values `included` or `excluded`. It specifies whether the OID subtree specified in <*subtree*> should be included or excluded from the corresponding view.
- <*mask*> is a list of hex octets, separated by '`.`' or '`:`'. The mask defaults to all `1`s (matching is performed against all digits in the subtree) if it is not specified.

A bit value of `0` in the mask acts as a wildcard for the corresponding value in the OID. A bit value of `1` in the mask indicates that the corresponding value in the subtree OID is fixed.

The mask enables you to control access to one row in a table in a relatively simple way.

```
view cust1 included 1.3.6.1.2.1.2.2.1.1.1 ff.a0
view cust2 included 1.3.6.1.2.1.2.2.1.1.2 ff.a0
```

In the example above, the hex value `ff.a0` has a binary equivalent of `11111111.10100000`. The bit position of the first '`0`' is position 10, which means that the view `cust1` provides access to all the objects with OIDs that match `1.3.6.1.2.1.2.2.1.x.1`, where `x` is an integer value.

Similarly, the view `cust2` provides access to all the objects with OIDs that match `1.3.6.1.2.1.2.2.1.x.2`.

VACM Examples

CODE EXAMPLE 3-2 VACM Example

```
#     sec.name  source   community
com2sec local  localhost  private
com2sec mynet  10.10.10.0/24  public
com2sec public  default  public

#    group.name sec.model sec.name
group mygroup v1     mynet
group mygroup v2c     mynet
group mygroup usm     mynet
group local  v1     local
group local  v2c     local
group local  usm     local
group public v1     public
group public v2c     public
group public usm     public

#   view.name incl/excl subtree mask
view all  included .1       80
view system included system      fe

# group.name context sec.model sec.level prefix read  write notify
access mygroup ""    any    noauth  exact mib2  none none
access public ""    any    noauth  exact system none none
access local  ""    any    noauth  exact all  all  all
```

Default VACM Model

The default configuration of the agent, as shipped, is functionally equivalent to the following entries:

CODE EXAMPLE 3-3 Default VACM Model

```
com2sec public  default  public
group public  v1  public
group public  v2c public
group public  usm public
view all included .1
access public  ""  any noauth  exact all none none
```

SNMPv3 Configuration

This section describes the command that enables the agent to respond to SNM_v3 messages

Syntax

```
engineID <string>
```

You must configure the `snmpd` agent with an `engineID` so that it can respond to SNMPv3 messages. If you do not configure an `engineID`, the agent uses a default value of the `engineID` based on the first IP address found for the hostname of the machine. If this is inappropriate (for example, if another agent is also using the same `engineID`) you must configure an `engineID` explicitly. This line configures the `engineID` to the value of <*string*>.

```
createUser <username> MD5 <authpassphrase>
```

`MD5` enables the authentication of SNMP users. When you create an SNMPv3 user, you must also update the VACM access control tables to provide an explicit declaration of what the user can access.

Note – This entry is made in `/var/opt/SUNWmasf/snmpd.dat` rather than `/etc/opt/SUNWmasf/conf/snmpd.conf`.

The minimum pass phrase length is eight characters.

Note – When the specified user has been created, the `createUser` statement provided in the file is removed.

Setting System Information

This section describes the commands that you use to configure the system information in MIB-II.

Syntax

```
syslocation <string>

syscontact <string>

sysname <string>
```

These parameters set the system location, system contact, or system name for the agent. This information is reported in the `system` group the MIB-II tree. Normally, these objects (`sysLocation.0`, `sysContact.0` and `sysName.0`) are read-write. However, if you specify the value for one of these objects by giving the appropriate token, the corresponding object becomes read-only, and subsequent attempts to set the value of the object result in a `notWritable` error response.

```
sysservices <number>
```

This parameter sets the value of the `system.sysServices.0` object. For a host, an acceptable value is 72.

The value of `sysservices` is a sum based on the network layers for which the node performs transactions. For each layer, L, in the range 1 through 7, for which this node performs transactions, 2 raised to (L - 1) is added to the sum. For example, a node that performs primarily routing functions would have a value of 4 (2^(3-1)). In contrast, a node which is a host offering application services would have a value of 72 (2^(4-1) + 2^(7-1)).

Note – In the context of the Internet suite of protocols, values should be calculated accordingly.

TABLE 3-1 provides these layer values.

TABLE 3-1 Internet Protocol Layer Functionality

layer	functionality
1	Physical (for example, repeaters)
2	Datalink/subnetwork (for example, bridges)
3	Internet (for example, IP gateways)
4	End-to-end (for example, IP hosts)
7	Applications (for example, mail relays)

For systems including OSI protocols, layers 5 and 6 can also be counted.

General Configuration

This section describes the commands that enable general configuration of the agent.

Syntax

```
agentgroup <groupid>
```

Change to this *groupid* after opening the port. The *groupid* can refer to a group by name, or a number if the group number starts with `#`. For example, specifying `agentgroup snmp` causes the agent to run as the `snmp` group, and specifying `agentgroup #10` causes the agent to run as the group with `groupid 10`.

```
agentuser <uid>
```

Change to this *uid* after opening the port. The *uid* can refer to a user by name, or a number if the user number starts with `#`. For example, specifying `agentuser snmp` causes the agent to run as the snmp user, and specifying `agentuser #10` causes the agent to run as the user with *uid* `10`.

```
authtrapenable <number>
```

Setting `authtrapenable` to `1` enables generation of authentication failure traps. The default value is `disabled(2)`. Normally, the corresponding object (`snmpEnableAuthenTraps.0`) is read-write, but setting its value with this token makes the object read-only, and subsequent attempts to set the value of the object result in a `notWritable` error response.

Updating the SNMP Agent When It Is Running

You can update the configuration file at any time, either before starting the agent or once the agent has started. If the agent is running, it does not update its running configuration unless explicitly requested to do so.

▼ To Force the SNMP Agent to Update

- **Type:**

```
# kill -HUP <pid>
```

Co-Existence With Other Agents

The SNMP agent must be configured to use a specific network port and this cannot be shared with any other component on the system. However, master agent technologies can enable multiple SNMP agents on a system to appear as a single entity to management applications. To use such solutions, configure this agent to use a port that is known to the master agent, which then presents an aggregated OID space.

CHAPTER 4

Troubleshooting

The Sun SNMP Management Agent has been designed so that it can provide full SNMP capabilities without further need for complex configuration. This chapter provides additional information should you experience problems using the agent.

The chapter contains the following sections:

Problems Starting or Accessing the Agent

If the agent does not appear to start, ensure that you are the root user before proceeding.

▼ To Check Whether the Agent is Running

1. **Type;:**

```
# ps -ef | grep snmpd
```

2. **Look for a line of the form:**

```
root 29394 1 0Feb 18 ?4:11 /opt/SUNWsmasf/sbin/snmpd
```

   If no such line appears, the agent is not running.

3. **Attempt to start the agent with the command:**

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

4. **Again, confirm whether the agent is now running. If the agent continues not to run, refer to the file** `/var/adm/messages`**.**

   Use your preferred editor to review the messages at the end of the file. Messages from the SNMP agent have the form:

```
date time  hostname snmpd[pid]: [ID id daemon.type] Text
```

   where:

   - `date` is the date when the message was created
   - `time` is the time of the message
   - `hostname` shows the host where the message was created
   - `pid` is the process id of the process that resulted in the message
   - `id` is an identifier

- `type` indicates the type of message, which can be `error`, `info`, or `warning`, depending on the nature of the problem being reported.

When reviewing the entries in the log file, make sure you confirm that the messages you are reviewing did result from the attempt to start the agent. If you are in any doubt, review the file before starting the agent and identify the date and time of the last message. After attempting to start the agent, locate this same message and review messages following it.

The most likely reason for the agent being unable to start is that it cannot access the network ports that it requires. The ports used by the agent are specified in the configuration file. Such a problem can be recognized by a message of the form:

```
date time  hostname snmpd[pid]: [ID id daemon.type] Error opening
specified endpoint "udp:portno"
```

where `portno` is the port specified in the configuration file.

If this message appears, identify why the port is not available. Use `netstat -a` to show all ports being accessed. If the port is not available, modify the configuration file to select a different port using an entry of the form:

```
agentaddress newport
```

If the error appears again for a port that is available, confirm you are running the agent as root and that you have permission to access the specified port.

If the management applications/clients cannot access agent, onfirm the agent has started (see "To Check Whether the Agent is Running" on page 25).

▼ To Confirm the Agent has Started

1. **From a client that is unable to access the agent using SNMP, use `ping` to make sure that the host running the SNMP agent is accessible over the network.**

   If the ping fails, you need to address the underlying network connectivity issues.

2. **Make sure that the client has the necessary access permissions for the SNMP agent. If you are using SNMPv1 or SNMPv2c, make sure that the community string specified by the client is the string expected by the agent.**

   If you are using SNMPv3, make sure that passwords for authentication have been correctly specified.

Failure to Receive Traps

The most likely reason for not receiving traps is failure to specify the trap destination for the SNMP agent. Confirm which version of trap the management application is expecting then look at the file `/etc/opt/SUNWmasf/conf/snmpd.conf`. For an SNMPv1 trap, there must be a line of the form:

```
trapsink <hostname>
```

where <*hostname*> is either the name of the host or the IP address of the destination. For an SNMPv2 trap, the entry should read:

```
trap2sink <hostname>
```

From the system running the SNMP agent, use `ping` to confirm that the chosen destination can be accessed over the network:

```
# ping <hostname>
```

If the ping fails, it is necessary to confirm network connectivity before taking any further steps.

If traps are still not received, confirm whether the management application is able to receive traps. If the management application has received traps from other hosts, no further action is necessary to confirm this. However, if this is the first time the management application has been used to receive traps, confirm that it is correctly listening for SNMP traps on port 162 or other appropriate port. As this is a low port number, the application that is to receive traps must run with root privileges. If the application is running, use `netstat` to confirm that it is listening:

```
UDP: IPv3
    Local Address           Remote Address    State
----------------------- ------------------ --------
       *.162                                  Idle
```

Remember, traps are not sent continuously and are sent only in the following circumstances:

- SNMP agent is starting or stopping.

- Potential problem has been detected by the SNMP agent (or the problem has been cleared).
- SNMP agent has detected a change to the hardware (objects being added or removed from the management model).
- Property value has changed.

The agent sends traps for all value changes except those that change rapidly, such as voltage readings, fan tachometers and temperature. For these kinds of environmental sensors, traps are sent only when thresholds are crossed.

If you are not sure if a trap has been sent, stop and restart the agent to generate cold start traps using:

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

followed by:

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

CHAPTER 5

Uninstalling the Software

Generally, all that is required to uninstall SNMP is to use the `pkgrm` command to remove the packages you installed. This procedure removes all the relevant files and links.

▼ To Uninstall the Software

Caution – If you have also installed Sun Management Center, remove only the SNMP-specific packages `SUNWmasfr` and `SUNWmasf`.

- **To remove the Sun Fire V125/V210, V215, V240, V245, and Netra 210/240 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWescpl SUNWescdl
```

- **To remove the Sun Fire V250 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWescfl SUNWescdl
```

- **To remove the Sun Fire V440/V445 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWeschl SUNWescdl
```

- **To remove the Netra 440 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWescnl SUNWescdl
```

- **To remove the Sun Fire V1280/E2900, or Netra 1280/1290 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf
```

- **To remove the Sun Fire T1000 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWeserl SUNWespdl SUNWescdl
```

- **To remove the Sun Fire T2000, Sun Blade T6300,and Sun SPARC Enterprise T5120/T5220 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWesonl SUNWespdl SUNWescdl
```

- **To remove the Netra T2000 platform agent packages from the platform agent server, type:**

```
# pkgrm SUNWmasfr SUNWmasf SUNWesonl SUNWespdl SUNWescdl
```

CHAPTER 6

Platform-Specific Information

The chapter contains the following sections:

For related platform documentation see "Supported Platforms" in the *Sun SNMP Management Agent Release Notes, Version 1.5.3*.

The `sunPlat` Class Definitions

This section contains platform-specific information relating to class definitions and attributes.

Equipment

sunPlatEquipmentAdministrativeState

The *sunPlatEquipmentAdministrativeState* is read-only and reports `unlocked(2)`, except in the case of network interfaces for which it reports `locked(1)` until the interface has been appropriately configured.

sunPlatEquipmentUnknownStatus

For all slots, bays, and PCI cards, the precise operation state cannot be determined from available sensors or be derived from other components. Therefore, these all report *sunPlatEquipmentUnknownStatus* as `true(1)`.

sunPlatEquipmentLocationName

For all objects this provides a string of the form:

```
<serial number>/<hostname>/<NAC Name>
```

where `<serial number>` is the serial number of the chassis. The `<NAC Name>` provides a unique name for each hardware component.

sunPlatEquipmentHolderPowered

Only read accesses are supported for this object.

Replaceable and Hotswappable Components

The following attributes are used to indicate a replaceable or hot-swappable component:

- *sunPlatCircuitPackReplaceable*
- *sunPlatCircuitPackHotSwappable*
- *isFRU*

A component is replaceable if it can be removed and replaced with a physically different component.

A component is hotswappable if the power can be *on* when the component is removed or inserted. All hotswappable components are replaceable.

A component is a Field Replaceable Unit (FRU) if it is an identifiable part that can be obtained and replaced in the field.

Binary Sensors

Changes in binary sensors that result in *sunPlatBinarySensorCurrent* not taking the same value as that specified by *sunPlatBinarySensorExpected* generate an alarm with a *sunPlatAlarmPerceivedSeverity* of `major(3)`. For example, when unplugging a cable from a power supply on the Sun Fire V240, an alarm is sent with a *sunPlatAlarmPerceivedSeverity* of `major(3)` and the value of *sunPlatEquipmentAlarmStatus* is set to `major(2)`.

Numeric Sensors

The following table shows the mappings that exist between the configured threshold, the severity reported in the alarm (*sunPlatAlarmPerceivedSeverity*) and the severity (*sunPlatEquipmentAlarmStatus*) recorded against the related object for all numeric sensors.

TABLE 6-1 Numeric Sensor Mapping

Threshold	sunPlatAlarmPerceivedSeverity	sunPlatEquipmentAlarmStatus
`lowerThresholdNonCritical(0)`	`warning(5)`	`warning(5)`
`upperThresholdNonCritical(1)`	`warning(5)`	`warning(5)`
`lowerThresholdCritical(2)`	`major(3)`	`major(2)`
`upperThresholdCritical(3)`	`major(3)`	`major(2)`
`lowerThresholdFatal(4)`	`critical(2)`	`critical(1)`
`upperThresholdFatal(5)`	`critical(2)`	`critical(1)`

For numeric sensors, the following objects are not instrumented:

- *sunPlatNumericSensorNormalMin*
- *sunPlatNumericSensorNormalMax*
- *sunPlatNumericSensorAccuracy*
- *sunPlatNumericSensorHysteresis*
- *sunPlatNumericSensorThresholdResetAction*

Note – *sunPlatNumericSensorEnabledThresholds* is read only.

Fan Tachometer Thresholds

The only threshold for fan tachometers configured is the `lowerThresholdNonCritical(0)`. When the detected speed of the fan falls below the configured non-critical threshold, an alarm is raised and the *sunPlatEquipmentAlarmStatus* is assigned a severity of `warning(5)`. As no further thresholds are set, no additional alarms are raised even if the speed of the fan drops further or stops.

Update Intervals

The agent initiates an update of its internal representation of the managed environment every 15 seconds.

Note – For the Sun Fire V1280/E2900 and Netra 1280/1290 systems, this interval is set for 10 seconds.

Logical and Log Table Population

In the current version of the agent, the following tables are not completely populated:

- `sunPlatWatchdogTable`
- `sunPlatLogTable`

  For consistency, this table has entries where:

  - *sunPlatLogAdministrativeState* is `locked(1)`
  - *sunPlatLogOperationalState* is `disabled(1)`
  - *RowStatus* is `notInService(2)`

  to indicate it is not in use.

- `sunPlatLogRecordTable`
- `sunPlatLogRecordAdditionalTable`
- `sunPlatLogRecordAlarmTable`
- `sunPlatLogRecordChangeTable`
- `sunPlatLogicalTable`
- `sunPlatUnitaryComputerSystemTable`
- `sunPlatInitialLoadInfoTable`

Additionally, for the Sun Fire V1280/E2900 and Netra 1280/1290 servers, the following tables are not completely populated:

- `sunPlatEquipmentHolderTable`
- `sunPlatCircuitPackTable`

CHAPTER 7

Introduction to SNMP

This chapter provides a brief introduction to the essential features of the Simple Network Management Protocol (SNMP). This chapter addresses the issues that are of particular relevance to the Sun Fire and Netra systems.

The chapter contains the following sections:

SNMP Versions

SNMP is an open internet standard for managing networked devices (systems). It is defined, in common with other internet standards, by a number of Requests for Comments (RFCs) published by the Internet Engineering Task Force (IETF).

There are three versions of SNMP that define approved standards:

- SNMPv1
- SNMPv2 (also known, and referred to in this document, as SNMPv2c)
- SNMPv3

SNMPv1 was first defined in 1988. SNMPv2 was introduced in 1993 and attempted to address some of the shortcomings of SNMPv1 by adding further protocol operations and data types and providing security. Limitations in the security model led to what is now accepted as the SNMPv2c standard, which dropped the new security-based features. Experimental versions, known as SNMPv2usec and SNMPv2*, also appeared at this time, but these have not been widely adopted and remain experimental.

SNMPv3, introduced in 1999, defines the SNMP management framework supporting pluggable components, including security.

For further information about these standards, refer to the following RFCs at the IETF website (`http://www.ietf.org/rfc.html`):

- SNMPv1: RFC1155, RFC1157, RFC1212, RFC1215
- SNMPv2: RFC2578, RFC2579, RFC2580, RFC3416
- SNMPv3: RFC3410, RFC3411, RFC3412, RFC3413, RFC3414, RFC3415
- Coexistence between standards: RFC2576

SNMP Managers and Agents

SNMP is a network protocol that allows devices to be managed remotely by a network management station (NMS), also commonly called a *manager*.

To be managed, a device must have an SNMP *agent* associated with it. The purpose of the agent is to:

- Receive requests for data representing the state of the device from the manager, and provide an appropriate response
- Accept data from the manager to enable control of the device state
- Generate SNMP *traps*, which are unsolicited messages sent to one or more selected mangers to signal significant events relating to the device

SNMP Management Information Base

To manage and monitor a device, its characteristics must be represented using a format known to both the agent and the manager. These characteristics can represent physical properties such as fan speeds, or services such as routing tables. The data structure defining these characteristics is known as a *management information base* (MIB). This data model is typically organized into tables, but can also include simple values. An example of the former is a routing table, and an example of the latter is a timestamp indicating the time at which the agent was started.

The MIB is a definition for a virtual data store accessible via SNMP. The content is accessible from the manager using `get` and `set` operations as follows:

- In response to a `get` operation, the agent provides data, either maintained locally or directly from the managed device.
- In response to a `set` operation, the agent typically performs some action affecting the state of either itself or the managed device.

To enable an NMS to manage a device via its agent, the MIB corresponding to the data presented by the agent must be loaded into the manager. The mechanism for doing this varies depending on the implementation of the network management software. This gives the manager the information required to address and interpret correctly the data model presented by the agent.

Note – MIBs can reference definitions in other MIBs, so to use a given MIB, it mightbe necessary to load others.

To address the content of this virtual data store, the MIB is defined in terms of *object identifiers* (OIDs). An OID consists of an hierarchically arranged sequence of integers that defines a unique name space. Each assigned integer has an associated text name. For example, the OID `1.3.6.1` corresponds to the OID name `iso.org.dod.internet` and `1.3.6.1.4` corresponds to the OID name `iso.org.dod.internet.private`.

The numeric form is used within SNMP protocol transactions, whereas the text form is used in user interfaces to aid readability. Objects represented by such OIDs are commonly referred to by the last component of their name as a shorthand form. To avoid confusion arising from this convention, it is normal to apply a MIB-specific prefix, such as *sunPlat*, to all object names defined therein, thereby making all such identifiers globally unique.

Note – The MIB is defined using a language known as ASN.1, the discussion of which is beyond the scope of this document. For reference, the structure of the MIBs for SNMPv2c is defined by its Structure of Management Information (SMI), defined in RFC2578. This defines the syntax and basic data types available to MIBs. The textual conventions (type definitions) defined in RFC2579 define additional data types and enumerations.

MIB Tables

Much of the data content defined by MIBs is in tabular form, and organized as entries consisting of a sequence of objects, each with its own OIDs. For example, a table of fan characteristics could consist of a number of rows, one per fan, with each row containing columns corresponding to the current speed, the expected speed, and the minimum acceptable speed.

The addressing of the rows within the table can be as follows:

- Simple, single-dimensional index (a row number within the table, for example '6')
- More complex, multidimensional, instance specifier such as an IP address and port number (for instance, `127.0.0.1, 1234`)

Each table definition within the MIB has an INDEX clause that defines which instance specifiers to use to select a given entry. In either case, the objects used to define the index to the required row must themselves be defined within the MIB. Thus, a table with a simple, single-dimensional index typically has an index column that is referenced by the table's INDEX clause. A specific data item within a table is then addressed by specifying the OID giving its columnar prefix.

For example, `myFanTable.myFanEntry.myCurrentFanSpeed`) with a suffix instance specifier (for instance `127.0.0.1.1234` from the previous example) gives `myFanTable.myFanEntry.myCurrentFanSpeed.127.0.0.1.1234`.

The SMI defining the MIB syntax provides an important capability for extending tables to add additional entries, effectively by adding extra columns to the table. This is achieved by defining a table with an INDEX clause that is a duplicate of the INDEX clause of the table being extended.

It is also possible to define MIB tables that are indexed not by objects contained within them, but by objects *imported* from other tables, potentially defined in other MIBs. This construct, effectively, enables columns to be added to an existing table.

Note – The `SUN-PLATFORM-MIB` makes extensive use of this mechanism to extend tables defined in the `ENTITY-MIB` (see Chapter 9).

Access Control

All addressable objects defined in the MIB have associated maximum access rights, for instance, *read-only* or *read-write*. These determine the maximum access the agent can support, and can be used by the manager to restrict the operations it will permit the operator to attempt. The agent is able to apply lower access rights as required, that is, it is able to refuse writes to objects that are considered read-write. This refusal can be on the basis of:

- How applicable the operation is to the object being addressed (for example, where an object defined by the MIB represents a state machine for which only certain transactions are legal)
- Security restrictions that limit certain operations to restricted sets of managers

The mechanism used to communicate security access rights in SMMPv1 is that of *community strings*. These are simply text strings such as *private* and *public* that are passed with each SNMP data request. As SNMPv1 and SNMPv2 requests are not encrypted, this should not be considered secure. The mechanism used to define which community strings the agent should respond to, and from which manager, depends on the implementation of the agent, but is typically based on access control lists (ACLs), which are files describing applicable access permissions.

For a description of how to configure ACLs, refer to Chapter 3.

CHAPTER 8

Platform Management Model

This chapter provides an overview of how SNMP models the hardware platform using the Sun platform SNMP model (SunPSM).

The chapter contains the following sections:

Modeling the Platform

The server is represented as a collection of nested *hardware resources* within a chassis. Some resources can be nested directly within the chassis, such as a motherboard. Others are nested within other resources—for example, a motherboard can include a processor. These relationships, extending from within the chassis, form a *hierarchy* of hardware resources, each physically contained within its enclosing *parent*. This hierarchy is modeled using *relationships* between *managed objects* that represent the hardware resources (FIGURE 8-1).

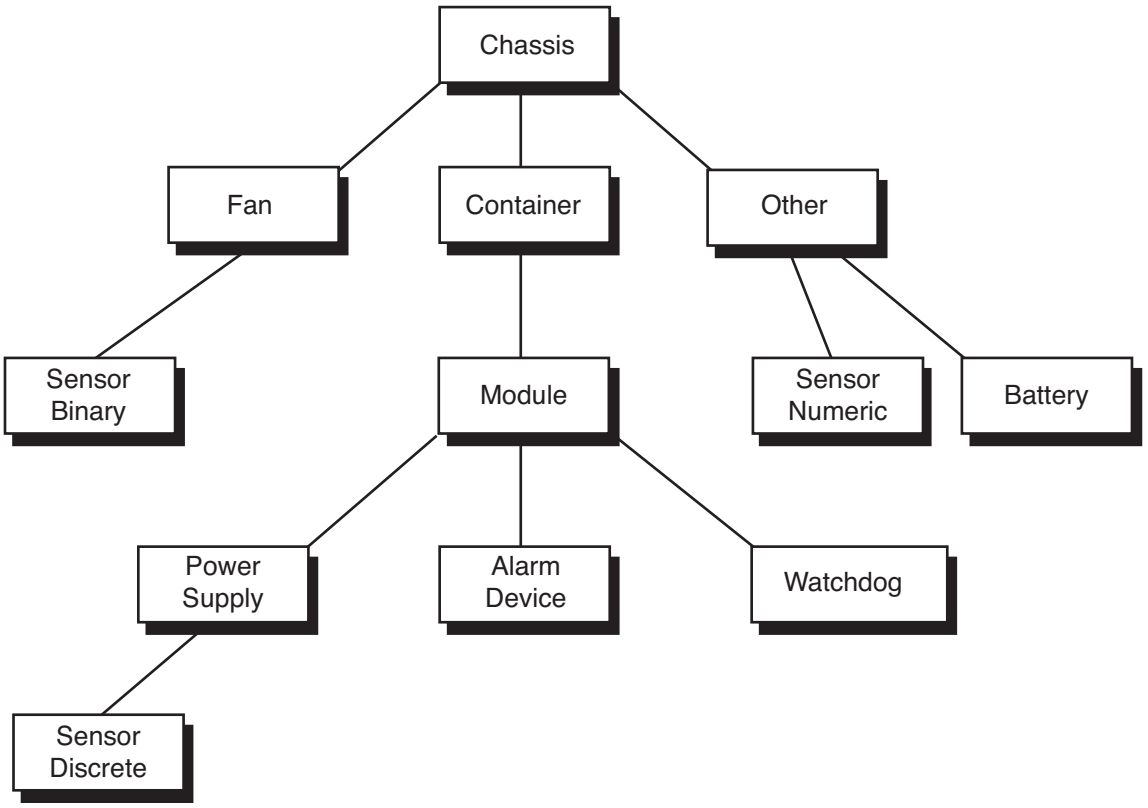

FIGURE 8-1 Hardware Resource Hierarchy Example

Managed Objects

The SunPSM model provides a useful set of common platform building blocks representing fundamental hardware resources. Instances of these platform building blocks are called *managed objects*. A hardware resource is represented by a managed object if it can be monitored or if it provides useful configuration information.

Additional managed objects are used to represent other features of the management interface. For example, hardware resources can issue asynchronous status reports, (*notifications*), in response to problems (*alarms*) or changes in configuration (*events*).

Managed objects are defined in terms of managed object *classes*. Characteristics of the resource are represented by *properties* of the managed object. New classes, called *subclasses*, are defined in terms of existing classes. A subclass *inherits* all the characteristics of its *superclass*, but represents its own characteristics by adding new properties.

FIGURE 8-2 shows the class inheritance hierarchy of the hardware building blocks defined by the SunPSM model.

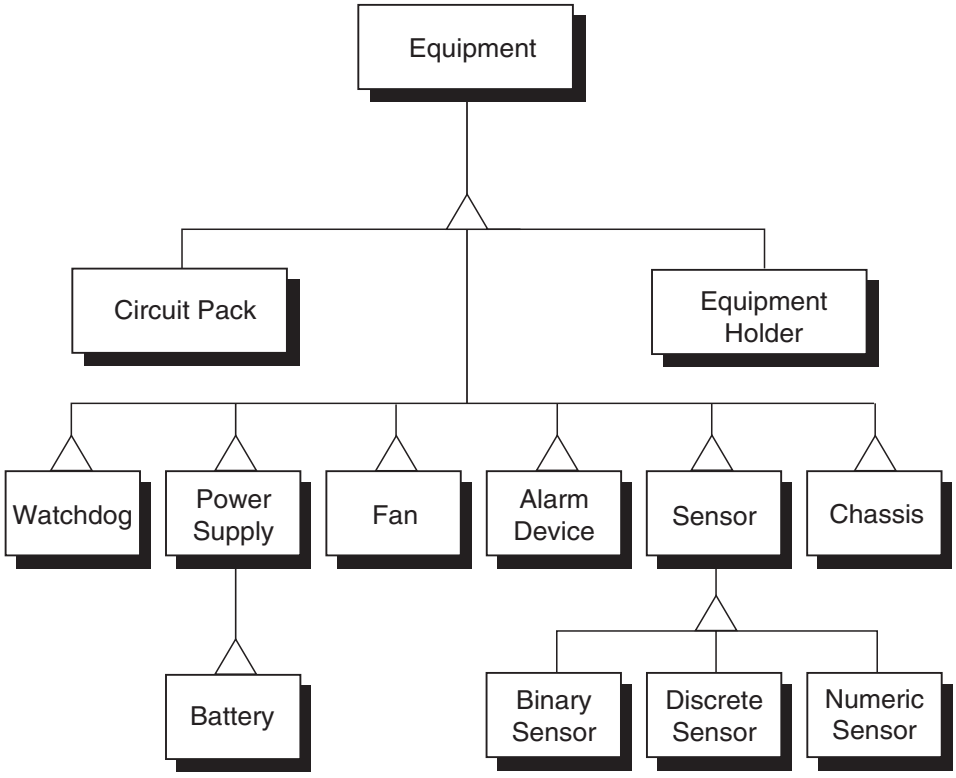

FIGURE 8-2 SunPSM Managed Object Class Inheritance Diagram

Derivation of `sunPlat` Classes

The SunPSM classes are based on industry-standard management concepts. The Sun SNMP management agent system uses a subset of the ITU-T generic network information model, chosen for its representation of hardware infrastructure. This provides a powerful and extendible framework that supports uniform fault and configuration management in a Telecommunications Management Network (TMN).

The Distributed Management Task Force (DMTF) common information model (CIM) schema models the physical environment, and event definition and handling, and provides system-specific extensions to the common model.

CHAPTER 9

Sun SNMP Management Agent MIBs

This chapter describes how the managed objects and their relationships are presented by the SNMP interface.

The chapter contains the following sections:

SNMP Representation of the Model

The SNMP agent supports both management agent system, and also a logical representation of the administrative domains within it, are provided by the ???, as defined by RFC 2737, extended by the SUN-PLATFORM-MIB.

Note – Many of the objects defined in the MIBs have a MAX-ACCESS of read-write, but these objects are only writable where such an operation is appropriate to the component being modeled.

The contains the following groups, which describe the physical and logical elements of the managed system.

entityPhysical Group

The entityPhysical group describes the physical entities—identifiable physical resources managed by the agent (for example, chassis, power supplies, sensors and so forth). These entities are represented by rows in the entPhysicalTable.

entityLogical Group[1]

The entityLogical group describes the logical entities managed by the agent. These are representations of logical entities providing abstractions of service that can be managed by higher levels of management. These are primarily concerned with platform hardware management and include functions such as OS reboot, hardware reset and power control. Typically, they correspond to administrative domains such as Solaris domains or service controllers.

entityMapping Group[1]

The entityMapping group identifies the relationship between the entityPhysical group and the entityLogical group.

entityGeneral Group

The entityGeneral group provides the last change time stamp for the time when any entity in the Physical Entity Table or Physical Mapping Table is changed.

entityMIBTraps Group

The entityMIBTraps group defines the entConfigChange notifications used to signal a change to the last change time stamp.

Chapter 7 provides an overview of how the generic elements of SNMP represent the Sun platform SNMP model.

1. These groups are not currently populated.

Physical Model

The SunPSM physical model uses the to provide a containment hierarchy of hardware entities. Each entity is modeled as a separate row of the ′s `entPhysicalTable`.

FIGURE 9-1 shows an example of a physical containment hierarchy. The number in the bottom right corner gives the index to the corresponding row in the `entPhysicalTable` (TABLE 9-1).

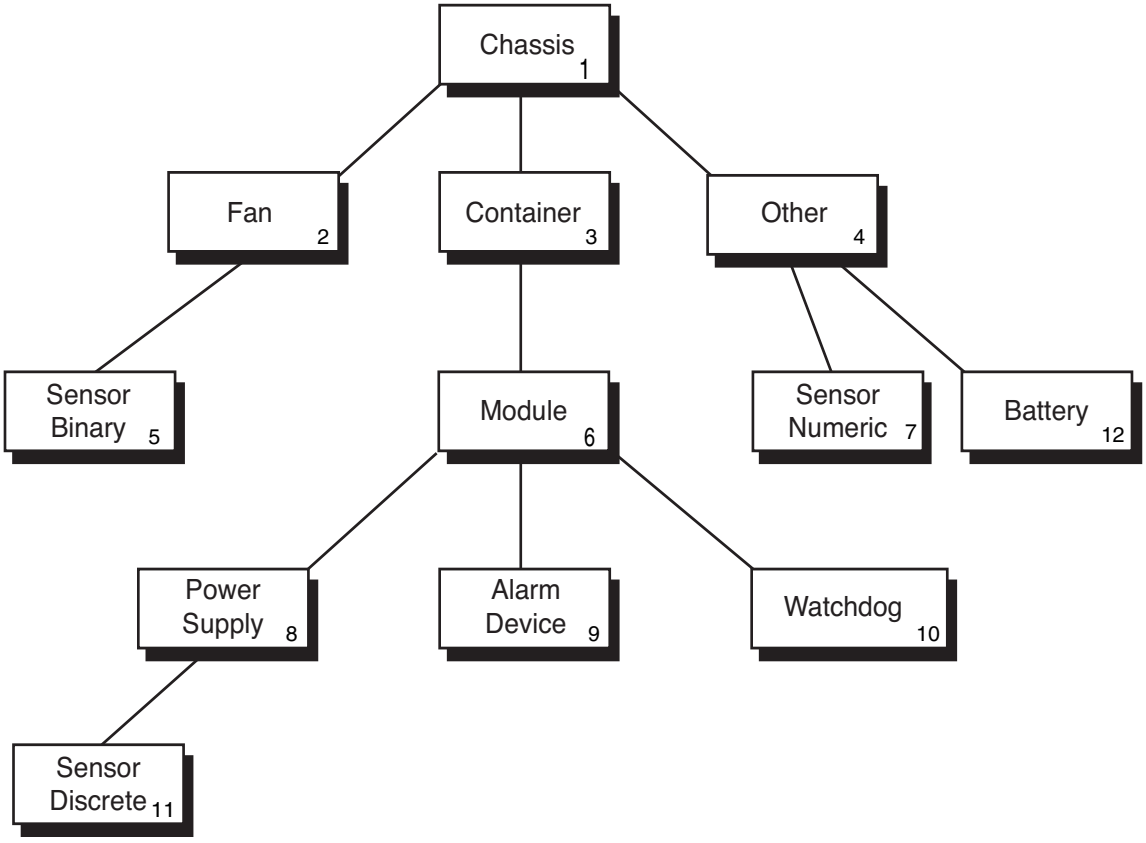

FIGURE 9-1 Physical Containment Hierarchy Example

This information is presented using SNMP tables:

- Physical entity table (`entPhysicalTable`)

  This table provides a row per hardware entity. These rows are called *Entries* and a particular row is referred to as an *instance*. Each entry contains:

  - Physical class (`entPhysicalClass`)
  - Common characteristics of the hardware entity
  - Unique index (`entPhysicalIndex`)
  - Reference (`entPhysicalContainedIn`) that points to the row of the hardware entity that acts as the *container* for this resource. This is zero for components, such as a chassis, that are not physically contained within another container.

- Physical Mapping Table (`entPhysicalContainsTable`)

  This table provides a virtual copy of the hierarchy of the hardware resources represented in the Physical Entity Table. This table is two-dimensional, indexed first by the `entPhysicalIndex` of the containing entry, and then by the `entPhysicalIndex` of each contained entry.

TABLE 9-1 shows the `entPhysicalTable` on which the above figure is based, and TABLE 9-2 shows the physical mapping.

TABLE 9-1 Physical Entity Table

entPhysicalIndex	entPhysicalClass	entPhysicalContainedIn
1	Chassis	0
2	Fan	1
3	Container (for example, a slot containing a FRU)	1
4	Other	1
5	Sensor (binary)	2
6	Module (for example, a pluggable FRU)	3
7	Sensor (numeric)	4
8	Power supply	6
9	Alarm device	6
10	Watchdog	6
11	Sensor (discrete)	8
12	Power supply (battery)	4

TABLE 9-2 Physical Mapping Table

entPhysicalIndex	entPhysicalChildIndex
1	2
1	3
1	4
2	5
3	6
4	7
4	12
6	8
6	9
6	10
8	11

Classes

The `entPhysicalClass` is an enumerated value that provides an indication of the general hardware type of a particular physical entity, each of which is represented by a row in the `entPhysicalTable`.

The following list describes the set of `entPhysicalClass` elements:

- `other(1)`

  The enumeration `other(1)` applies if the physical entity cannot be classified by one of the following.

- `chassis(3)`

  The `chassis` class represents an overall container for equipment. Any class of physical entity can be contained within a chassis.

- `container(5)`

  The `container` class applies to a physical entity that can contain one or more removable physical entities, of the same or different type. For example, each empty or full slot in a chassis is modeled as a container. Field-replaceable units (FRUs), such as a power supply or fan, are modeled as modules within a container entity.

- `powerSupply(6)`

  The `power supply` class applies to a component that can supply power, such as a battery.

- `fan(7)`

  The `fan` class applies if the physical entity is a fan or other cooling device.

- `sensor(8)`

  The `sensor` class applies to a physical entity that is capable of measuring some physical property.

- `module(9)`

  The `module` class applies to a self-contained sub-system, and which is modeled within another physical entity such as a `chassis` or another `module`. The entity is always modeled within a `container`

Logical Model

Note – The logical model is not populated by this agent.

The `sunPlat` logical model uses the to provide a list of logical entities. Each entity is represented as a separate row in the 's `entLogicalTable`.

Note – Unlike the physical model, the logical model is flat rather than hierarchical.

The does not distinguish between different classes of logical object, unlike the case for physical objects. The `SUN-PLATFORM-MIB` provides a class hierarchy for logical objects and this is described in Chapter 8.

The information in the `entLogicalTable` can be used to support multi-scoping using different naming context. However, this capability is not employed in this product. The information of particular value is the `entLogicalDescription` and `entLogicalTAddress`, the latter giving the IP address at which the logical entity can be accessed.

Logical and Physical Hierarchy Mapping

The provides a mapping between logical objects and the physical objects of which they are composed. This is achieved by the `entLPMappingTable`, which is a two-dimensional table (similar to the `entPhysicalContainsTable`) and which identifies the physical entities that realize a given logical entity. These physical entities are identified by their `entLPPhysicalIndex`, which is equivalent to the `entPhysicalIndex`.

Although this table can potentially represent all the physical entities associated with a given logical entity, by convention, only the enclosing physical entity is referenced. For example, for a logical entity realized by a physical module, the mapping references only the module, not all the physical entities contained within it.

Event and Alarm Model

The provides a single SNMP notification, `entConfigChange`, which is used to signal a change to any of the tables in the MIB. It is set to provide a maximum of one trap every five seconds.

The `SUN-PLATFORM-MIB` defines more specific notifications and these are described in Chapter 11.

The `SUN-PLATFORM-MIB`

The `SUN-PLATFORM-MIB` does the following:

- Extends the Physical Entity Table to represent new classes of component
- Extends the Logical Entity Table to represent high value platform and server objects

---

Note – All objects in the `SUN-PLATFORM-MIB` have the prefix `sunPlat` to make them globally unique.

Physical Model Table Extensions

The SUN-PLATFORM-MIB provides additional attributes from classes that are not represented in the Physical Entity Table. It extends the Physical Entity Table by adding the following sparsely populated table extensions:

- Equipment Table Extension

  This augments the Physical Entity Table to provide further information for managed objects of the Equipment class. This class is applicable for all Sun SNMP Management Agent hardware resources. Subclasses of the Equipment class are represented by further Table Extensions.

  - Equipment Holder Table Extension

    This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `container(5)` `entPhysicalClass`.

  - Circuit Pack Table Extension

    This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `module(9)` `entPhysicalClass`.

- Physical Table Extension

  This extends the Physical Entity Table. It is used to supplement the `entPhysicalClass` column in the Physical Entity Table. If a resource has an `entPhysicalClass` of `other(1)`, but is of a class modeled by `sunPlat`, that is, the Watchdog or AlarmDevice class, this table identifies its `sunPlatPhysicalClass`.

- Sensor Table Extension

  This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `entPhysicalClass` `sensor(8)`. Subclasses of Sensor class are represented by further Table Extensions and identified by this table using `sunPlatSensorClass`.

  - Binary Sensor Table Extension

    This extends the Sensor Table Extension. It provides additional information relevant for managed objects of the `entPhysicalClass` `sensor(8)` and `sunPlatSensorClass` `binary(1)`.

  - Numeric Sensor Table Extension

    This extends the Sensor Table Extension. It provides additional information relevant for managed objects of the `entPhysicalClass` `sensor(8)` and `sunPlatSensorClass` `numeric(2)`.

- Discrete Sensor Table Extension

  This extends the Sensor Table Extension. It provides additional information relevant for managed objects of the `entPhysicalClass sensor(8)` and `sunPlatSensorClass discrete(3)`.

- Fan Table Extension

  This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `entPhysicalClass fan(7)`.

- Alarm Table Extension

  This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `entPhysicalClass other(1)` and `sunPlatPhysicalClass alarm(8)`.

- Watchdog Table Extension

  This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `entPhysicalClass other(1)` and `sunPlatPhysicalClass watchdog(3)`, typically representing service indicator LEDs.

- Power Supply Table Extension

  This extends the Equipment Table Extension. It provides the additional information relevant for managed objects of the `entPhysicalClass powerSupply(6)`.

TABLE 9-3 shows an example of the Table Extensions to the Physical Entity Table. The `entPhysicalIndex` (column 1 in this table) is based on the example hardware resource hierarchy shown in FIGURE 9-1

TABLE 9-3 Physical Entity Table Extensions

ENTITY-MIB			SUN-PLATFORM-MIB											
entPhysicalIndex	entPhysicalClass				sunPlatPhysicalClass		sunPlatFanClass		sunPlatSensorClass					sunPlatPowerSupplyClass
1	chassis													
3	container													
8	power supply													G
12	power supply													H
2	fan						A							
	fan						B							
	fan						C							
5	sensor								D					
7	sensor								E					
11	sensor								F					
6	module													
9	other				alarm									
10	other				watchdog									
4	other				other									
entPhysicalTable		sunPlatEquipmentTable	sunPlatEquipmentHolderTable	sunPlatCircuitPackTable	sunPlatPhysicalTable	sunPlatWatchdogTable	sunPlatFanTable	sunPlatAlarmTable	sunPlatSensorTable	sunPlatBinarySensorTable	sunPlatNumericSensorTable	sunPlatDiscreteSensorTable	sunPlatDiscreteSensorStatusTable	sunPlatPowerSupplyTable

TABLE 9-4 Key to Physical Entity Table Extensions

Reference	Description
A	Fan
B	Refrigeration
C	Heat sink
D	Binary
E	Numeric
F	Discrete
G	Power supply
H	Battery

Logical Model Table Extension

The `SUN-PLATFORM-MIB` provides additional attributes from classes that are not supported in the Logical Entity Table. It extends the Logical Entity Table by adding the following sparsely populated table extension:

- Logical Class Table Extension

  This extends the `entLogicalTable` to define the class of the logical entity, `sunPlatLogicalClass`, and its status, `sunPlatLogicalStatus`. The `sunPlatLogicalTable` is valid for all entries in the `entLogicalTable`. The Computer System subclass of the Logical class is represented by a further table extension:

  - Computer System Table Extension

    This table extends the `entLogicalTable` to provide attributes common to instances of a computer system.

    The `sunPlatUnitaryComputerSystemTable` is valid for those rows of the `entLogicalTable` with a `sunPlatLogicalClass` of `computerSystem(2)`.

CHAPTER 10

Physical Model

This chapter describes the sunPlat physical class hierarchy and how the managed physical object classes defined in the sunPlat model are represented by the SUN-PLATFORM-MIB.

The chapter contains the following sections:

The sunPlat Physical Class Hierarchy

FIGURE 10-1 shows the inheritance hierarchy of the sunPlat classes used to model hardware resources within the Sun SNMP Management Agent.

The Physical Entity superclass provides an attribute for defining the relationship between managed objects. It also provides standard SNMP attributes that correspond to attributes in the Equipment class.

The sunPlat Equipment class is derived from the Physical Entity superclass to provide the additional attributes defined in the corresponding classes that are applicable for fault monitoring.

The sunPlat Equipment Holder and sunPlat Circuit Pack classes are derived from the sunPlat Equipment superclass to represent receptacles and the components that plug into them, respectively.

The sunPlat Equipment class is then further specialized to provide the DMTF-derived classes.

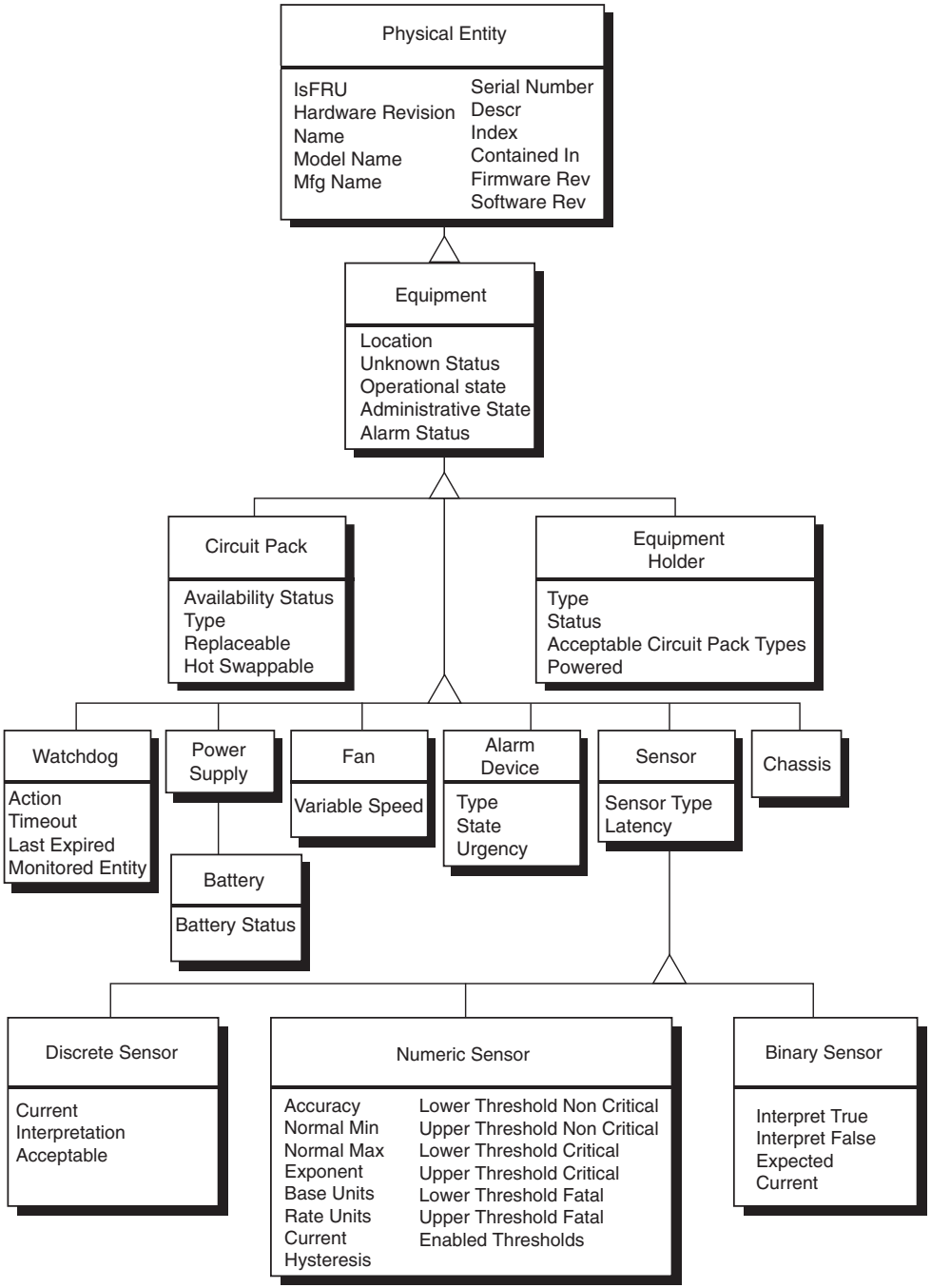

FIGURE 10-1 The `sunPlat` Physical Resource Inheritance Class Diagram

The sunPlat Class Definitions

The attributes of the sunPlat classes are used to represent the characteristics of hardware resources. The availability and operability of the resource to the manager are represented by the *state* of the managed object. Different sunPlat classes have a variety of attributes that express aspects of the managed object's state.

Physical Entity

The Physical Entity superclass is used to represent the characteristics that are generic to all resources.

Note – The *entPhysical* prefix has been omitted from the following attribute names for clarity.

- Descr

  This is a text string containing the known name for the resource. This name is typically the name used to describe the resource in product documentation, on product legends or, possibly, the name stored in firmware.

- IsFRU

  This is a boolean representing whether the resource is a field replaceable unit. Only hardware resources of the class sunPlatCircuitPack are considered to be FRUs.

- HardwareRev

  This is a text string containing the manufacturer's hardware revision information for the resource. Not all hardware resources have associated hardware revision information.

- Name

  This is a text string containing the logical name by which the resource is known to the operating system and associated utilities. This name can be a device node or a defined name used by system utilities, where applicable. Not all resources have a device name.

- ModelName

  This is a text string containing the manufacturer's customer visible part number or part definition. Not all hardware resources have associated part numbers or definitions.

- SerialNum

This is a text string containing the manufacturer's serial number for the resource. Not all hardware resources have associated serial numbers.

- `MfgName`

  This is a text string containing the manufacturer's name for the resource. Not all hardware resources have an associated manufacturer's name.

The Physical Entity superclass also contains attributes that are used for describing the hierarchy of hardware resources:

- `Class`

  This enumerated type contains an indication of the general hardware type of a particular physical resource. The supported values of this class are defined by the `ENTITY-MIB`. This attribute can be used as an indication of the relevant Table Extensions for the managed object. The mapping between the `ENTITY-MIB` classes and the `sunPlat` classes are as shown in TABLE 10-1:

TABLE 10-1 Physical Entity Superclass 'Class' Attribute Mapping

entPhysicalClass	sunPlat Class
chassis(3)	`sunPlatChassis`
backplane(4)	Not implemented
container(5)	`sunPlatEquipmentHolder`
powerSupply(6)	`sunPlatPowerSupply`
fan(7)	`sunPlatFan`
sensor(8)	`sunPlatSensor`, plus subclasses
module(9)	`sunPlatCircuitPack`
port(10)	Not implemented
stack(11)	Not implemented
other(1)	`sunPlatEquipment`, plus subclasses
unknown(2)	Not implemented

- `Index`

  This integer uniquely identifies the entry in the Physical Entity Table that identifies the managed object. Values are not pre-allocated and might vary on each invocation of the agent.

- `ContainedIn`

  This integer represents the *Index* attribute of the managed object containing this managed object. The attribute therefore models the relationship between the managed objects.

Note – The object at the root of the physical containment hierarchy (typically a chassis) is not physically contained within another entity represented in the table. To indicate this, its *entPhysicalContainedIn* value is set to `0`.

- `FirmwareRev`

  This is a text string containing the manufacturer's firmware revision information for the resource. Not all hardware resources have associated firmware revision information.

- `SoftwareRev`

  This is a text string containing the manufacturer's software revision information for the resource. Not all hardware resources have associated software revision information.

The `sunPlat` Equipment Class

The `sunPlat` Equipment class is used to represent the characteristics that are generic to all hardware resources. This class contains attributes representing configuration and generic health status information. This class is further subclassed to provide more detail configuration information and monitoring data for particular types of resources.

The `entPhysicalClass` is dependent on the subclass being represented.

The `sunPlat` Equipment class has the following attributes:

Note – The `sunPlatEquipment` prefix has been omitted from the following attribute names for clarity.

- `AdministrativeState`

  This read-write attribute takes one of the following enumerated values representing the current administrative state of the resource:

  - `locked(1)`
  - `unlocked(2)`
  - `shuttingDown(3)`

- `OperationalState`

  This read-only attribute is an enumerated type indicating whether the resource is physically installed and capable of providing service. The attribute contributes to the *state* of the managed object and can take the values shown in TABLE 10-2.

TABLE 10-2 Operational State Attribute Values

Attribute Values	Description
`disabled(1)`	The resource is totally inoperable and unable to provide service to the user.
`enabled(2)`	The resource is partially or fully operable and available for use.

- `AlarmStatus`

  This read-only attribute takes an enumerated value representing the current alarm state of the resource. It indicates the highest severity of any alarm outstanding on the managed object. The attribute can take the following values:

  - `critical(1)`
  - `major(2)`
  - `minor(3)`
  - `indeterminate(4)`
  - `warning(5)`
  - `pending(6)`
  - `cleared(7)`

- `UnknownStatus`

  This read-only attribute indicates if the other state attributes might not reflect the true state of the resource. The attribute takes a boolean value representing whether the managed object is able to report accurately faults against the resource. If the resource is unable, truthfully, to reflect its *state*, this attribute is set to `true`.

- `LocationName`

  This read-only attribute contains a locator for the resource. For resources contained directly within the chassis, this attribute correlates with legends on slots and product documentation, or provides a geographical indication of the position of the resource within the chassis. Other hardware resources typically have a *location* corresponding to the *Name* of the managed object for the resource in which it is contained.

The `sunPlat` Circuit Pack Class

The `sunPlat` Circuit Pack class is used to represent the characteristics that are generic to a replaceable resource or FRU. A replaceable resource is defined as a hardware module whose purpose is to package internal hardware components into a recognized form-factor. Typically, a FRU will have a defined form-factor and

physical appearance. It can be a pluggable removable unit, which is plugged into a connector, it can be more permanently sited within a bay, or it can fit into a drawer, rack or shelf.

This class has the `entPhysicalClass` `module(9)`.

The `sunPlat` Circuit Pack class has the following attributes:

Note – The `sunPlatCircuitPack` prefix has been omitted from the following attribute names for clarity.

- `Type`

  This read-only attribute is a text string used for assessing the resource's compatibility with its container. This attribute can identify functionality and form-factor characteristics of the resource.

- `AvailabilityStatus`

  This read-only attribute further qualifies the *Operational State* of the managed object. It is an object using BITS syntax, and can take zero or more of the set of values shown in TABLE 10-3. Not all of these are applicable to every class of managed object. This attribute contributes to the *state* of the managed object.

TABLE 10-3 Availability Status Attribute Values

Attribute Values	Bit No.	Hex.	Description
`inTest(0)`	`0`	`80`	The resource is undergoing a test procedure.
`failed(1)`	`1`	`40`	The resource has an internal fault that prevents it from operating. *Operational State* is `disabled(1)`.
`powerOff(2)`	`2`	`20`	The resource requires power to be applied and is not powered on.
`offLine(3)`	`3`	`10`	The resource requires a routine operation to be performed to place it online and make it available for use. *Operational State* is `disabled(1)`.
`offDuty(4)`	`4`	`08`	The resource has been made inactive by an internal control process.

TABLE 10-3 Availability Status Attribute Values *(Continued)*

Attribute Values	Bit No.	Hex.	Description
`dependency(5)`	5	04	The resource cannot operate because some other resource on which it depends is unavailable. *Operational State* is `disabled(1)`.
`degraded(6)`	6	02	The service available from the resource is degraded in some respect, such as in speed or operating capability. However, the resource remains available for service. *Operational State* is `enabled(2)`.
`notInstalled(7)`	7	01	The resource represented by the managed object is not present, or is incomplete. *Operational State* is `disabled(1)`.

- `Replaceable`

  This read-only attribute takes a boolean value indicating whether the resource is a replaceable unit.

- `HotSwappable`

  This read-only attribute takes a boolean value indicating whether the replaceable resource is hot swappable.

The `sunPlat` Equipment Holder

The `sunPlat` Equipment Holder class is used to represent the characteristics of hardware resources that are capable of holding removable hardware resources.

This class has the `entPhysicalClass container(5)`.

The `sunPlat` Equipment Holder class has the following attributes:

Note – The `sunPlatEquipmentHolder` prefix has been omitted from the following attribute names for clarity.

- `Type`

  This read-only attribute is an enumerated type representing the holder type of the resource, as shown in TABLE 10-4:

TABLE 10-4 Equipment Holder Type Attribute Values

Attribute Values	Description
`bay(1)`	A bay is typically a unit of vertical space within a rack that contains shelves or drawers for holding telecommunications equipment. The `sunPlat` class interprets its use within a chassis as a physical receptacle requiring cables for signal connections
`shelf(2)`	A horizontal support or subrack for holding telecommunications equipment within a rack.
`drawer(3)`	A horizontal enclosure for holding telecommunications equipment within a rack.
`slot(4)`	A physical receptacle with an integral connector for signal connections for removable equipment.
`rack(5)`	A rack is the support infrastructure for holding telecommunications equipment, holders, and cable management systems within a self-contained enclosure.

- `AcceptableTypes`

  This read-only attribute is a list of text strings representing the types of removable resource (circuit pack) that are supported by the holder. These types are tested for compatibility with the removable resource's *Type* attribute.

- `Status`

  This read-only attribute is an enumerated type indicating the status of the holder with regards to any replaceable hardware resources (circuit packs) that it may contain, as shown in TABLE 10-5.

TABLE 10-5 Equipment Holder Status Attribute Values

Attribute Values	Description
`holderEmpty(1)`	There is no removable resource in the holder.
`inTheAcceptableList(2)`	The holder contains a removable resource that is one of the types in the *AcceptableTypes* list.
`notInTheAcceptableList(3)`	The holder contains a removable resource recognizable by the network element; but not one of the types in the *AcceptableTypes* list.
`unknownType(4)`	The holder contains an unrecognizable removable resource.

- ***Powered***

  This read-write attribute is an enumerated type indicating the power state of the resource. The possible values are:

  - `other(1)`
  - `unknown(2)`
  - `powerOff(3)`
  - `powerOn(4)`

The `sunPlat` Power Supply

The `sunPlat` Power Supply class is used to represent a power supply. It does not extend the characteristics of the `sunPlat` Equipment class. A power supply typically contains sensors representing monitored properties, for example voltages, current, and temperature. It can also contain other hardware resources such as fans. This is modeled using relationships between the managed objects.

If a power supply is a removable resource, it is modeled within a managed object of `sunPlat` Circuit Pack class.

This class has the `entPhysicalClass` `powerSupply(6)`.

The `sunPlat` Power Supply class has the following attribute:

Note – The `sunPlatPowerSupply` prefix has been omitted from the following attribute name for clarity.

- Class

  This read-only attribute is an enumerated type indicating the class of the power supply, and takes the following values:

  - other(1)
  - powerSupply(2)
  - battery(3)

The sunPlat Battery

The sunPlat Battery class is used to represent a power supply that supplies power from a battery.

This class has the entPhysicalClass powerSupply(6) and the sunPlatPowerSupplyClass battery(3).

The sunPlat Battery class has the following attribute:

Note – The sunPlatBattery prefix has been omitted from the following attribute name for clarity.

- Status

  This read-only attribute is an enumerated type that indicates the status of the battery, and takes the following values:

  - other(1)
  - unknown(2)
  - fullyCharged(3)
  - low(4)
  - critical(5)
  - charging(6)
  - chargingAndHigh(7)
  - chargingAndLow(8)
  - chargingAndCritical(9)
  - undefined(10)
  - partiallyCharged(11)

The sunPlat Watchdog

The sunPlat Watchdog class is used to represent the characteristics of timer hardware resources that allow the hardware to monitor the state of the operating system or applications.

This class has the `entPhysicalClass other(1)` and the `sunPlatPhysicalClass watchdog(3)`.

The `sunPlat` Watchdog class has the following attributes:

Note – The `sunPlatWatchdog` prefix has been omitted from the following attribute names for clarity.

- `Timeout`

  This read-only attribute is an integer indicating the interval in milliseconds after which the watchdog will timeout if not reset.

- `Action`

  This read-only attribute is an enumerated type representing the action taken by the watchdog if it is not reset within the period specified by the *Timeout*. The possible values are shown in TABLE 10-6.

TABLE 10-6 Watchdog Action Attribute Values

Action	Description
`statusOnly(1)`	The watchdog is readable by software, but performs no action.
`systemInterrupt(2)`	The watchdog generates a hardware interrupt to the system being monitored.
`systemReset(3)`	The watchdog resets the system being monitored.
`systemPowerOff(4)`	The watchdog powers off the system being monitored.
`systemPowerCycle(5)`	The watchdog powers off, and then on, the system being monitored.

- `LastExpired`

  This read-only attribute indicates the date and time at which the watchdog last expired.

- `MonitoredEntity`

  This read-only attribute is an enumerated type representing the entities that can be monitored by the watchdog. The possible values are:

  - `unknown(1)`
  - `other(2)`
  - `operatingSystem(3)`
  - `operatingSystemBootProcess(4)`
  - `operatingSystemShutdownProcess(5)`
  - `firmwareBootProcess(6)`
  - `biosBootProcess(7)`

- application(8)
- serviceProcessors(9)

The sunPlat Alarm

The sunPlat Alarm class is used to represent the characteristics of hardware resources that emit indications relating to problem situations, for instance buzzers, LEDs, relays, vibrators, and software alarms.

This class has the entPhsicalClass other(1) and the sunPlatPhysicalClass alarm(2).

The sunPlat Alarm class has the following attributes:

Note – The sunPlatAlarm prefix has been omitted from the following attribute names for clarity.

- Type

  This read-only attribute is an enumerated type representing the means by which the alarm condition is communicated. The possible values are shown in TABLE 10-7.

TABLE 10-7 Alarm Type Attribute Values

Attribute Values	Description
other(1)	The alarm device type is not one of the following.
audible(2)	The alarm device is audible change on the device.
visible(3)	The alarm causes a visible change on the device.
motion(4)	The alarm causes motion of the device.
switch(5)	The alarm causes an electrical signal change.

- State

  This read-write attribute is an enumerated type representing the state of the alarm The possible values are shown in TABLE 10-8.

TABLE 10-8 Alarm State Attribute Values

Attribute Values	Description
`unknown(1)`	The state of the alarm is undefined or unobservable.
`off(2)`	The alarm is inactive.
`steady(3)`	The alarm is active.
`alternating(4)`	The alarm is cycling between its inactive and active states.

- `Urgency`

  This read-write attribute is an enumerated type indicating the relative frequency at which the Alarm flashes, vibrates and/or emits audible tones. The possible values are:

  - `other(1)`
  - `unknown(2)`
  - `notSupported(3)`
  - `informational(4)`
  - `nonCritical(5)`
  - `critical(6)`
  - `unrecoverable(7)`

The `sunPlat` Fan

The `sunPlat` Fan class is used to represent the characteristics of active cooling devices. A fan typically contains a sensor representing the speed of rotation. This is modeled using a physical containment relationship between the `sunPlat` Fan managed object and a tachometer-managed object of class `sunPlatSensor`.

This class has the `entPhysicalClass fan(7)`.

The `sunPlat` Fan class has the following attribute:

Note – The `sunPlatFan` prefix has been omitted from the following attribute name for clarity.

- `Class`

  This read-only attribute is an enumerated type indicating the class of cooling device, and takes the following values:

  - `other(1)`
  - `fan(2)`
  - `refrigeration(3)`

- heatPipe(4)

The sunPlat Sensor

The sunPlat Sensor superclass is used to represent the generic characteristics of hardware resources that measure properties of other hardware resources.

This class has the entPhysicalClass sensor(8).

The sunPlat Sensor class has the following attributes:

Note – The sunPlatSensor prefix has been omitted from the following attribute names for clarity.

- Class

  This read-only attribute is an enumerated type indicating the class of the sensor, and takes the following values:

  - binary(1)
  - numeric(2)
  - discrete(3)

- Type

  This read-only attribute is an enumerated type identifying the property that the sensor measures. Some of the possible values of *Type* are shown in TABLE 10-9.

TABLE 10-9 Sensor Type Attribute Values

Type	Description
temperature(3)	A sensor for measuring the environmental temperature.
voltage(4)	A sensor for measuring the electrical voltage.
current(5)	A sensor for measuring the electrical current.
tachometer(6)	A sensor for measuring the speed/revolutions of a device.
counter(7)	A general purpose sensor which counts defined events.

- Latency

  This read-only attribute indicates the following:

  - Where the sensor is polled, this integer represents the update interval measured in milliseconds.
  - Where the sensor is event-driven, this value represents the maximum expected latency in processing that event.

The sunPlat Binary Sensor

A sunPlat Binary Sensor class is used to represent the characteristics of sensors that return binary output. It augments the sunPlatSensor table to provide the attributes that are specific to binary sensors.

This class has the entPhysicalClass sensor(8) and the sunPlatSensorClass binary(1).

The sunPlat Binary Sensor class has the following attributes:

Note – The sunPlatBinarySensor prefix has been omitted from the following attribute names for clarity.

- Current

  This read-only attribute takes a boolean value indicating the most recent value of the sensor.

- Expected

  This read-only attribute takes a boolean value indicating the anticipated value of the sensor.

- InterpretTrue

  This read-only attribute is a text string indicating the interpretation of a true value from the sensor.

- InterpretFalse

  This read-only attribute is a text string indicating the interpretation of a false value from the sensor.

The sunPlat Numeric Sensor

A sunPlat Numeric Sensor class is used to represent the characteristics of sensors which can return numeric readings. The numeric sensor values are qualified by a Unit of Measurement as defined below:

Unit of Measurement = *Base Unit* * $10^{Exponent}$

This qualification allows for units of measurement such as milliamps and microvolts. If a *Rate Unit* is defined, the Unit of Measurement is further refined as below:

Unit of Measurement = *Base Unit* * $10^{Exponent}$ per *Rate Unit*

This qualification allows for units of measurement such as rpm and km/hr.

This class has the `entPhysicalClass sensor(8)` and the `sunPlatSensorClass numeric`(2).

The `sunPlat` Numeric Sensor class has the following attributes:

Note – The `sunPlatNumericSensor` prefix has been omitted from the following attribute names for clarity.

- `BaseUnits`

  This read-only attribute is an enumerated type indicating the unit of measurement, prior to qualification as defined above. Examples of values of this type are:

  - `degC(3)`
  - `volts(6)`
  - `amps(7)`

- `Exponent`

  This read-only attribute is an integer that used to scale the *Base Unit* by some power of 10. For example, if *sunPlatNumericSensorBaseUnits* is set to `volts` and *sunPlatNumericSensorExponent* is set to `-6`, the units of the values returned are `microVolts`.

- `RateUnits`

  This read-only attribute is an enumerated type that indicates whether the sensor is measuring an absolute value (when the value is `none`) or a rate. In the latter case, the unit specified in *sunPlatNumericSensorBaseUnits* is expressed as 'per unit of time'. For example, if *sunPlatNumericSensorBaseUnits* is set to `degC` and *sunPlatNumericSensorRateUnits* is set to `perSecond`, the value represented has the units `degC/second`.

  Examples of values of this type are:

  - `perMicrosecond(2)`
  - `perMillisecond(3)`
  - `perSecond(4)`
  - `perMinute(5)`
  - `perHour(6)`
  - `none(1)`

- `Current`

  This read-only attribute is an integer indicating the most recent value of the sensor.

- `NormalMin`

This read-only attribute is an integer indicating the defined threshold below which the sensor reading is not expected to fall. This value is expressed in terms of the units of measurement as defined above. The attribute may not be applicable to some sensors.

- `NormalMax`

  This read-only attribute is an integer indicating the defined threshold above which the sensor reading is not expected to rise. This value is expressed in terms of the units of measurement as defined above. The attribute may not be applicable to some sensors.

- `Accuracy`

  This read-only attribute is an integer indicating the degree of error of the sensor for the measured property as a percentage to two decimal places. The value can vary depending on whether the sensor reading is linear over its dynamic range.

- `LowerNonCriticalThreshold`

  This read-only attribute is an integer indicating the lower threshold at which a `nonCritical` condition occurs.

- `UpperNonCriticalThreshold`

  This read-only attribute is an integer indicating the upper threshold at which a `nonCritical` condition occurs.

- `LowerCriticalThreshold`

  This read-only attribute is an integer indicating the lower threshold at which a `critical` condition occurs.

- `UpperCriticalThreshold`

  This read-only attribute is an integer indicating the upper threshold at which a `critical` condition occurs.

- `LowerFatalThreshold`

  This read-only attribute is an integer indicating the lower threshold at which a `fatal` condition occurs.

- `UpperFatalThreshold`

  This read-only attribute is an integer indicating the upper threshold at which a `fatal` condition occurs.

- `Hysteresis`

  This read-only attribute describes the hysteresis around the threshold values.

- `EnabledThresholds`

  This is read-only attribute that, when written to, resets the sensors to their default values.

The sunPlat Discrete Sensor

The sunPlat Discrete Sensor class is used for sensors that cannot be represented by the sunPlat Numeric Sensor or sunPlat Binary Sensor classes

This class has the entPhysicalClass sensor(8) and the sunPlatSensorClass discrete(3).

The class comprises two tables. The sunPlatDiscreteSensorTable has one attribute, sunPlatDiscreteSensorCurrent, which indicates the current state of the sensor expressed as an index in the sunPlatDiscreteSensorStatesTable.

The sunPlat Discrete Sensor class has the following attributes:

Note – The sunPlatDiscreteSensorState prefix has been omitted from the following attribute names for clarity.

- Index

  This read-only attribute takes a number that represents the index of a row in the sunPlatDiscreteSensorStatesTable, which identifies this sensor state.

- Interpretation

  This read-only attribute is a string describing the state represented by the corresponding row of the sunPlatDiscreteSensorStatesTable.

- Acceptable

  This read-only attribute takes a boolean value that indicates whether the state represented by this row of the table is considered acceptable.

The sunPlat Chassis

The sunPlat Chassis class is used to represent the primary enclosure. It does not extend the characteristics of the sunPlat Equipment class. The chassis contains all the modeled hardware resources, and is not contained within any other resource.

This class has the entPhysicalClass chassis(3).

CHAPTER 11

Traps

This chapter describes the properties and function of traps.

The chapter contains the following sections:

Overview

The SNMP agent provides traps to provide asynchronous updates to management applications. The agent provides notifications for the following:

- Change of value within an object
- Removal of an object from the management model
- Addition of an object to the management model
- Raising of an error or warning condition against a component within the model
- Clearance of an error or warning condition

The SNMP agent determines whether an error or warning condition exists against an object based on any sensors related to that object and/or any sub components of that object. For example, a CPU can have an associated fan and the fan can have an associated tachometer. All three components have an ability to report their operational status. If the tachometer reports that it has a failed operational status, this relates only to the health of the tachometer.

However if the tachometer reports that the speed of the fan has fallen below a particular threshold, a trap is raised against the fan rather than the tachometer. At this point the CPU may be still be completely functional. If the CPU has a related

temperature sensor and the fan continues to fail to function, the temperature could rise. If, at some point, the temperature exceeds a threshold value, a fault is issued against the CPU.

Multiple thresholds can be set for all numeric sensors:

- `upperFatal`
- `upperCritical`
- `upperNonCritical`
- `lowerFatal`
- `lowerCritical`
- `lowerNonCritical`

The precise semantics of these thresholds depends on the underlying component and not all components use all the thresholds. Where a component uses a threshold, the agent always creates a trap when the threshold is crossed. This means that if the current value reported by the sensor changes dramatically and crosses multiple thresholds at the same time, multiple traps are delivered.

Note – Even if multiple thresholds are crossed in one sample, when the problem is cleared, the resultant traps can occur over a period of time.

You can determine the threshold values assigned to a numeric sensor by reading the following properties:

- `sunPlatNumericSensorLowerThresholdNonCritical`
- `sunPlatNumericSensorUpperThresholdNonCritical`
- `sunPlatNumericSensorLowerThresholdCritical`
- `sunPlatNumericSensorUpperThresholdCritical`
- `sunPlatNumericSensorLowerThresholdFatal`
- `sunPlatNumericSensorUpperThresholdFatal`

These are all 32-bit integer values, the actual numeric threshold being determined by applying the exponent value provided by `sunPlatNumericSensorExponent`.

For binary sensors, a trap is sent when the value reported by `sunPlatBinarySensorCurrent` is not the same as that specified by `sunPlatBinarySensorExpected`. A trap indicating the problem has cleared is sent when `sunPlatBinarySensorCurrent` returns to the same value as `sunPlatBinarySensorExpected`.

For any binary sensor, you can determine the precise semantics of the reported values from the descriptions supplied by `sunPlatBinarySensorInterpretTrue` and `sunPlatBinarySensorInterpretFalse`.

To configure the agent to send traps to a particular management application please see Chapter 3.

Feature Enhancement

To support more information in SNMP traps, a configuration file option can be set by the system administrator that controls the inclusion of an extra `varbind` in traps that initially did not include it in its `varbind` list.

The SNMP variable `sunPlatNotificationAdditionalText` is now included in the following traps:

- `sunPlatStateChange`
- `sunPlatAttributeChangeInteger`
- `sunPlatAttributeChangeString`
- `sunPlatAttributeChangeOID`

In previous versions of the `SUN-PLATFORM-MIB`, these traps did not include the `sunPlatNotificationAdditionalText`. The conventional value of this SNMP variable is the `NAC` name (a formal slash-delimited path, in ASCII, which describes the entity's position in the entity-tree). Having the `NAC` name in each of the `AttributeChange` or `StateChange` traps helps remote NMSs more directly identify the entity that is causing the trap to be sent.

To allow the system administrator to choose which mode to run in (the older `SUNWmasf` style traps or the new style), you must edit the `snmpd` configuration file. The file is located in `/etc/opt/SUNWmasf/conf/snmpd.conf`.

▼ To Edit the File to Run the Older `SUNWmasf` Mode

1. **Locate the line that has the following content:**

```
#SUNW_alwaysIncludeEntPhysName yes
```

2. **Leave the line commented out, by using a leading #, as follows:**

```
#SUNW_alwaysIncludeEntPhysName yes
```

▼ To Edit the File to Run the Newer Version

If you want to run in the new style of traps, where each of the `StateChange` and `AttributeChange` traps include the `NAC` name in their `sunPlatNotificationAdditionalText` variable, do the following.

1. **Uncomment the line by removing the # from the front of the line.**
2. **Save the file.**
3. **Exit.**

   The next time `SUNWmasf` starts, it uses the setting, which was saved in the config file and run in that mode, for traps.

Standard Trap Properties

The information communicated in traps generated by the agent is based on a common set of attributes. For clarity, the `sunPlatNotification` prefix has been omitted from the following attribute names:

- `EventId`

  This is an integer uniquely identifying the notification and an indication of the order in which the notifications were generated by the agent.

Note – The agent does not guarantee that its sequencing reflects the order of the underlying events from which the notifications were generated.

- `Time`

  This is a timestamp indicating the time at which the notification was generated.

- `Object`

  This attribute is an OID that provides a direct reference to an entry in the MIB representing the resource with which the event is associated.

- `CorrelatedNotifications`

  This attribute is comma-separated list of ID values that identify the other events to which this event is associated.

- `AdditionalInfo`

  This attribute provides an additional OID that further qualifies the purpose of the trap. Not all traps populate this value, and a value of 0.0 identifies that no trap specific value is relevant to the trap.

- `AdditionalText`

  This attribute provides a further textual description for the trap. The exact content of the string varies depending on the cause of the trap. However, the format used for this string is always of the form:

<serial number>/<hostname>/<*entPhysicalName*>:<trap specific text>

where entPhysicalName identifies the physical name of the object raising the trap.

Examples of the <trap specific text> include:

- Voltage threshold crossed
- Current threshold crossed
- Tachometer threshold crossed
- Temperature threshold

- PerceivedSeverity

  This attribute describes the severity of the cause of the trap. The attribute can take the following values:

  - indeterminate(1)
  - critical(2)
  - major(3)
  - minor(4)
  - warning(5)
  - cleared(6)

- ProbableCause

  This attribute provides a textual description of the probable cause.

  Example of the permitted values include:

  - lowTemperature
  - coolingSystemFailure
  - externalEquipmentFailure

- SpecificProblem

  This attribute provides an additional textual description that supplies further information about the cause of the trap.

- RepairAction

  This attribute provides a textual description of the potential repair actions. Multiple repair actions may be described, in which case <CR><LF> sequences are used to delimit individual descriptions of the actions.

- ChangedOID

  This attribute provides the OID of an object whose value changing caused the trap to be sent

Trap Types

This section describes the function and properties of the following trap types:

- Object creation and deletion traps
- Property change traps
- Environmental and status alarm traps

Object Creation and Deletion Traps

The `sunPlatObjectCreation` trap is sent when an object is created within the agent to represent a new component. For example, a new component has been hotplugged into the system. The agent sends object creation traps only for objects added once it has started, so traps are not generated during the initial phase of discovery when the agent first starts.

The `sunPlatObjectDeletion` trap is sent when an object is deleted as a result of a component being removed or unconfigured from the system. Object creation and deletion traps have the following properties:

Note – The `sunPlatNotification` prefix has been omitted from the following attribute names for clarity.

- `EventId`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `Time`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `Object`

  The attribute is OID of the parent of the object being created or deleted. The OID is that of the `entPhysicalDescr` object for the row in `entPhysicalTable` that represents the parent object.

- `CorrelatedNotifications`

  This is currently assigned to an empty string.

- `AdditionalInfo`

  This attribute is the OID of object being created. The OID is that of the `entPhysicalDescr` object for the row in `entPhysicalTable` being created.

- `AdditionalText`

  For details, see "Standard Trap Properties" on page 82.

Property Change Traps

The agent can deliver traps whenever a value in an object changes. The type of trap depends on the type of the object:

- The `sunPlatStateChange` trap is sent when a state attribute changes value.
- The `sunPlatAttributeChangeInteger` trap is sent when the value of an integer property changes. This trap is used for all integer property value changes, with the exception of current values with respect to numeric sensors as such values can change rapidly and can result in a large number of traps.
- The `sunPlatAttributeChangeString` trap is sent when the value of a string property changes. This trap is used for all string property value changes.
- The `sunPlatAttributeChangeOID` trap is sent when the value of an OID property changes. This trap is used for all OID property value changes.

Depending on the object type causing the trap, different attributes in the trap body are used to supply both before and after values of the changed attribute:

Note – The `sunPlatNotification` prefix has been omitted from the following attribute names for clarity.

- `OldInteger`

  This attribute is used for `sunPlatStateChange` and `sunPlatAttributeChangeInteger` traps. The attribute provides the original value of the object identified by `sunPlatNotificationChangedOID`.

- `NewInteger`

  This attribute is used for `sunPlatStateChange` and `sunPlatAttributeChangeInteger` traps. The attribute provides the new value of the object identified by `sunPlatNotificationChangedOID`.

- `OldString`

  This attribute is used by the `sunPlatAttributeChangeString` trap. This is the original value of the object identified by `sunPlatNotificationChangedOID`.

- `NewString`

  This attribute is used by the `sunPlatAttributeChangeString` trap. This is the new value of the object identified by `sunPlatNotificationChangedOID`.

- `OldOID`

  This attribute is used by the `sunPlatAttributeChangeOID` trap. This is the original value of the object identified by `sunPlatNotificationChangedOID`.

- `NewOID`

  This attribute is used by the `sunPlatAttributeChangeOID` trap. This is the new value of the object identified by `sunPlatNotificationChangedOID`.

Environmental and Status Alarm Traps

The agent sends traps to report potential environmental problems and other warning or error conditions. The definition of environmental, or other, condition depends on the component, but examples include the speed of a fan dropping below a pre-determined threshold and the temperature of a component rising above a threshold.

Sensors, such as numeric sensors, have multiple thresholds defined to reflect warning, critical and failure conditions. If a sensor value crosses multiple thresholds when sampled, traps are sent for all thresholds that have been crossed. Similarly, when the reading provided by the sensor returns to a value within an acceptable range, trap(s) are sent to indicate that the warning or error condition has cleared.

The traps used by the agent are:

- `sunPlatCommunicationsAlarm`
- `sunPlatEnvironmentalAlarm`
- `sunPlatEquipmentAlarm`
- `sunPlatProcessingErrorAlarm`

The trap used depends on the nature of the problem as defined by the generic network information model (ITU-T Recommendation M.3100). The attributes supplied by the traps are:

Note – The `sunPlatNotification` prefix has been omitted from the following attribute names for clarity.

- `EventId`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `Time`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `Object`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `CorrelatedNotifications`

This attribute is assigned an empty string when an alarm is first raised. When the alarm condition is cleared, the value is the string representation of the `sunPlatNotificationEventId` value that was sent in the trap when the alarm condition was first raised.

- `AdditionalInfo`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `AdditionalText`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `PerceivedSeverity`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `ProbableCause`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `SpecificProblem`

  This attribute is set as described in "Standard Trap Properties" on page 82.

- `RepairAction`

  This attribute is set as described in "Standard Trap Properties" on page 82.

Glossary

This glossary contains definitions of terminology, acronyms, and abbreviations for the Sun SNMP Management Agent for supported servers.

A

ACL access control list

C

CIM common information model

D

DAQ Data Acquisition, process of collecting data from a monitored server

DMTF Distributed Management Task Force

DVD Digital Versatile Disc

E

ENTITY-MIB Describes physical and logical entities

F

FRU ID Field Replaceable Unit Identifier

I

IETP Internet Engineering Task Force

IP Internet Protocol

ITU-T Internet Telecommunication Union Standardization Sector

L

LED light-emitting diode

M

Mbytes megabytes

MIB Management Information Base

N

NIM network information model

NMS network management station

O

OID Object Identifier

OS operating system

P

PCP Platform Channel Protocol, library framework that allows communication between the host OS and the system's service processor

PICL Platform Information and Control Library

R

RFC Request for Comments

S

SMA SNMP Management Agent

SMI Structure of Management Information

SNMP Simple Network Management Protocol

`SUN-PLATFORM-MIB` Extends the Entity MIB to provide additional information about hardware components

SunPSM Sun platform SNMP model

T

TMN Telecommunications Management Network

U

UDP User Diagram Protocol

V

VACM view-based access control model

Index

T

V

W