# Sun™ SNMP Management Agent Guide for the Sun Fire™ B1600

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-1010-10
March 2003, Revision 01

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

# Contents

# Figures

# Tables

# Code Samples

# Preface

This Guide describes the Sun SNMP Management Agent for the Sun Fire B1600 platform, which supports management of the platform hardware using the Simple Network Management Protocol.

The Management Agent provides *monitoring* of inventory, configuration, and environmental and fault reporting. It also provides *control* and *monitoring* of service indicators, and of power, standby and reset of the processor blades.

It is intended to be read by experienced Enterprise Administrators and professional developers.

The Guide is divided into two parts:

- Part 1 (Chapter 1 through Chapter 8) introduces the SNMP Management Agent and describes its functionality.
- Part 2 (Chapter 9 through Chapter 13) explains how to install and configure the software.

## How This Book Is Organized

The Guide contains the following chapters:

**Part 1**

**Chapter 1** describes the components of the Sun SNMP Management Agent software.

**Chapter 2** provides a brief introduction to the essential features of the Simple Network Management Protocol (SNMP).

**Chapter 3** describes the functionality and features of the SNMPv3 Master Agent.

**Chapter 4** provides an overview of how SNMP models the Sun Fire B1600.

**Chapter 5** describes how the Sun Fire B1600 managed objects and their relationships are presented by the SNMP interface.

**Chapter 6** 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 7** describes the sunPlat logical class hierarchy and how the managed object classes defined in the sunPlat model are represented by the SUN-PLATFORM-MIB.

**Chapter 8** describes the SunPlat notifications classes and attributes, as defined in the SUN-PLATFORM-MIB.

**Part 2**

**Chapter 9** describes the components that make up the management software for the Sun Fire B1600 and lists the system checks you should make before installing the SNMP software.

**Chapter 10** describes how to install the management software on the Sun Fire B1600.

**Chapter 11** provides information about the user configurable files.

**Chapter 12** describes the default configuration after installation, and explains how to modify the configuration files.

**Chapter 13** explains how to uninstall the software.

**Chapter 14** provides help in troubleshooting your software.

**Appendix A** describes how to install J2RE to co-exist with J2SE, and how to modify the startup scripts to locate the installation.

# Typographic Conventions

| Typeface or Symbol | Meaning | Examples |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories; on-screen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`% You have mail.` |
| **`AaBbCc123`** | What you type, when contrasted with on-screen computer output | `% su`<br>`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*.<br>These are called *class* options.<br>To delete a file, type **`rm`** *filename*. |

# Shell Prompts

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

# Related Documentation

| Application | Title | Part Number |
|---|---|---|
| SunMC | *SunMC 3.0 Supplement for the Sun Fire B1600* | 817-1011 |
| Release Notes | *SNMP Release Notes for the Sun Fire B1600* | 817-1006 |
| Sun Fire B1600 Platform | *Sun Fire B1600 Blade System Chassis Hardware Installation Guide* | 816-7614 |
| | *Sun Fire B1600 Blade System Chassis Software Setup Guide* | 816-3361 |
| | *Sun Fire B1600 Blade System Chassis Administration Guide* | 816-4765 |
| | *Sun Fire B1600 Blade System Chassis Switch Administration Guide* | 816-3365 |

# Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

`http://www.sun.com/documentation`

# Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

`http://www.sun.com/service/contacting`

# 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 Guide for Sun Fire B1600*, part number 817-1010-10

# PART I Technical Description and Functionality

CHAPTER 1

# Sun SNMP Management Agent Supplement

This release of the Sun™ SNMP Management Agent provides monitoring and control of the Sun Fire ™ B1600 shelf and Sun Fire B100s blade.

Depending on the platform type, you can employ:

- A domain agent, running on the Sun Fire B100s blade (domain hardware monitoring)

  The software is installed locally on the server being monitored and only that server can be monitored. In the case of the Sun Fire B1600, each blade is monitored separately.

  The scope of domain hardware monitoring for the Sun Fire B100s blade is limited to the hardware of the blade only. It does not include other shelf components such as the service indicators, PSUs, SSCs and the identity of the shelf itself.

- A platform agent, proxied through a system controller (platform hardware monitoring)

  The software is installed on a remote (platform agent) server that accesses platform instrumentation through the system controller. This enables you to monitor all the hardware managed by the system controller.

  The scope of platform hardware monitoring for a Sun Fire B1600 includes the shelf, its identity, service indicators, and all its field replaceable units (FRUs). In addition, some hardware information (specifically voltage monitoring) about the Sun Fire B100s blades is available that is not available using domain hardware monitoring.

  FIGURE 1-1 shows an example of both types of hardware monitoring. Sun Fire B1600 shelves A and B are connected to the network management station via a platform agent server (platform hardware monitoring). In the case of Sun Fire B1600 shelf C, the Sun Fire B100s blades are connected directly to the network management station (domain hardware monitoring).

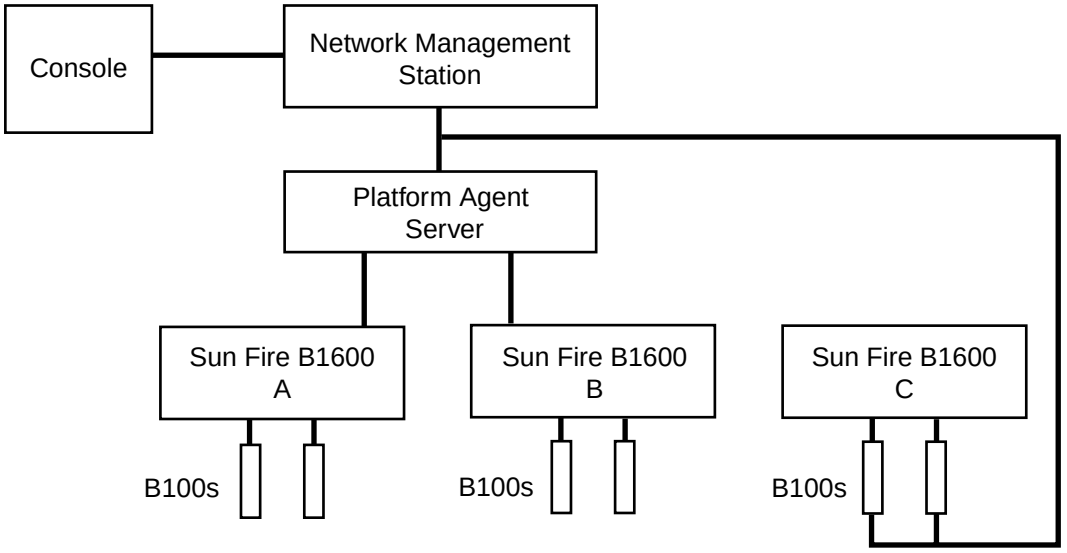


**FIGURE 1-1** Example of Domain and Platform Hardware Monitoring

The software comprises a number of packages that provide the following functionality:

- An SNMP sub-agent

  By default, the SNMP sub-agent is registered as sub-agent of the Solaris Master Agent, snmpdx. The sub-agent is also known as the *SNMP Mediator.*

- An SNMPv3 Master Agent

  The SNMPv3 Master Agent provides single, secure point of presence from which SNMP Mediators residing on the platform can be accessed. The Master Agent acts as a proxy by forwarding requests to `snmpdx`.

- Sun Fire B1600 and Sun Fire B100s instrumentation.

  These packages are installed as required, depending on whether domain-based or platform-based hardware monitoring is employed.

CHAPTER 2

# Introduction to SNMP

This chapter provides a brief introduction to the essential features of the Simple Network Management Protocol (SNMP). It is by no means exhaustive and addresses the issues that are of particular relevance to the Sun Fire™ B1600 system.

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 web site (`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* (known as the SNMP Mediator) associated with it. The purpose of the Mediator 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 Mediator 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 manger. 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 may be 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:

- A simple, single-dimensional index (a row number within the table, for example ‘6’)
- A more complex, multi-dimensional, 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 specifier(s) 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 5).

---

## 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 11.

# SNMP Master Agents

A manager communicates with the agent by sending (UDP) packets to a well known port (161) on the system on which the agent is running. If several agents are running on a given system, each managing different devices, there is a potential conflict for the use of the port resource. One possible solution is to use different, non-standard, ports numbers for each agent. An alternative is to introduce the concept of a *Master Agent*, which accepts SNMP requests on behalf of all the agents running on a given system and forwards these requests as appropriate. This has the benefit of allowing a manager to access all SNMP agents in a consistent manner. The Sun Fire B1600 supports either approach.

For further information about the Master Agent, refer to Chapter 3.

# SNMP Mediator and snmpdx

`snmpdx` is the standard Solaris™ SNMP agent and is an SNMPv1 master agent.

By default, the SNMP Mediator is registered as a sub-agent of `snmpdx`. In this configuration, only SNMPv1 `get` and `set` requests are supported although SNMPv2c notifications are issued.

The relationship between `snmpdx` and the SNMP Mediator is further developed in Chapter 10 and Chapter 11.

See also the man page `snmpdx`(1M).

CHAPTER 3

# Master Agent

This chapter describes the functionality and features of the SNMPv3 Master Agent.

The chapter contains the following sections:



## Functionality

The SNMPv3 Master Agent provides single, secure point of presence through which SNMP management information can be accessed.

The SNMPv3 Master Agent binds to the SNMP service port (default 161) and forwards all requests to `snmpdx`, the standard Solaris master agent, which in turn forwards these requests to the appropriate registered sub agents. `snmpdx` is supplied as part of the standard Solaris distribution, but it supports SNMPv1 only and therefore does not directly provide the security offered by SNMPv3. The Master Agent translates all requests, be they SNMPv1, v2c or v3, to SNMPv1, so that `snmpdx` can handle them.

In effect, Master Agent acts as an SNMP firewall by providing secure access to all existing sub agents.

# Configuration Overview

This section provides an introduction to the way in which you configure SNMP to include Master Agent functionality. The topic is developed in detail in Chapter 11, which includes a full description and examples of the configuration files referred to in the remainder of this section.

The SNMP Mediator is registered as a sub-agent of snmpdx using an automatically allocated port number.

---

**Note –** The SNMP Agent is referred to as the SNMP Mediator throughout this Guide.

---

When Master Agent is enabled, the `snmpdx` automatic start up is disabled and Master Agent is registered on port 161. A new port number is assigned to `snmpdx`, which is started under control of the Master Agent's startup file.

Configuration is by means of:

- The SNMPv3 security files `spama.uacl` and `spama.security`
- The SNMPv1/v2 access control list (ACL) file `spama.acl`
- The configuration file `spama.conf`
- The startup script, `spama`

It is not possible to configure Master Agent dynamically while it is running.

The SNMP Mediator also requires configuration and this is described in Chapter 11.

CHAPTER 4

# The Platform Management Model

This chapter provides an overview of how SNMP models the Sun Fire B1600 system using the Sun Platform SNMP Model (sunPlat).

The chapter contains the following sections:



## Modeling the Sun Fire B1600 Platform

The Sun Fire B1600 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.

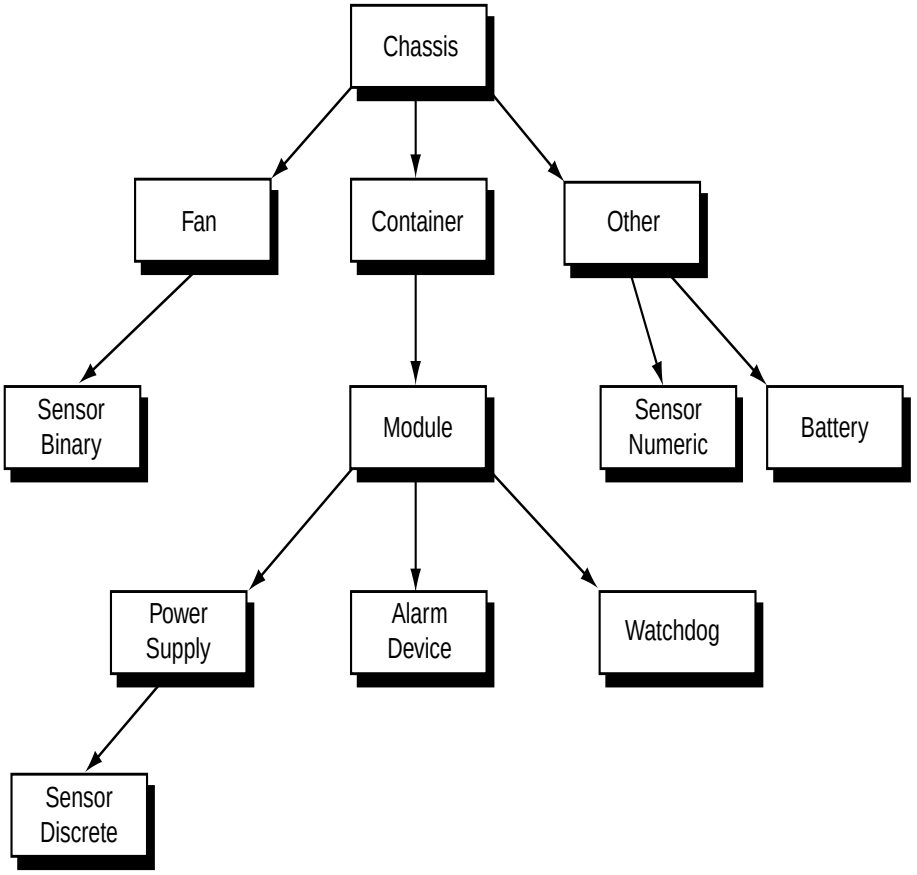


**FIGURE 4-1** Example Hardware Resource Hierarchy

# Managed Objects

The sunPlat 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 4-2 shows the class inheritance hierarchy of the hardware building blocks defined by the sunPlat model.

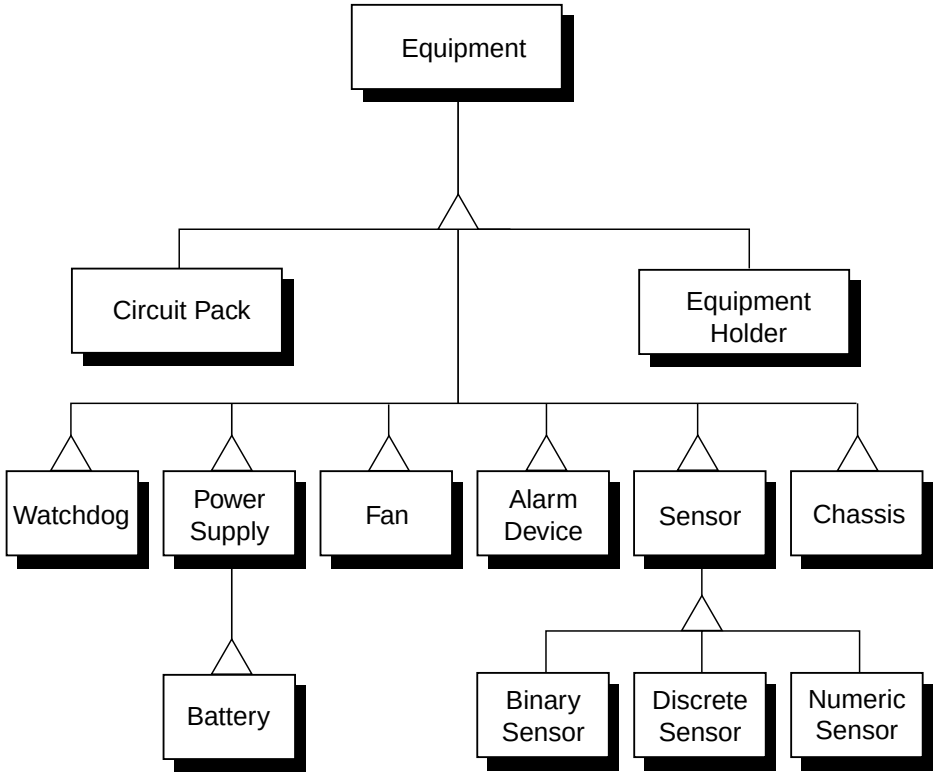


**FIGURE 4-2** sunPlat Managed Object Class Inheritance Diagram

# Derivation of sunPlat Classes

The sunPlat classes are based on industry-standard management concepts. The Sun Fire B1600 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 5

# The Sun Fire B1600 MIBs

This chapter describes how the Sun Fire B1600 managed objects and their relationships are presented by the SNMP interface.

The chapter contains the following sections:



## SNMP Representation of the Model

The SNMP Mediator supports both polled and event-based management. The physical components of the Sun Fire B1600 system and also a logical representation of the administrative domains within it are provided by the ENTITY-MIB, 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 ENTITY-MIB 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*

The `entityLogical` group describes the logical entities managed by the agent. These are representations of *high value* logical entities providing abstractions of service that must 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*

The `entityMapping` group identifies the relationship between the `entityPhysical` group and the `entityLogical` group. This function is handled internally by the SNMP Mediator.

### *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 *entPhysicalChange* notifications used to signal a change to any object in the ENTITY-MIB.

Chapter 2 provides an overview of how the generic elements of SNMP represent the Sun Platform SNMP Model.

# The Physical Model

The sunPlat physical model uses the ENTITY-MIB to provide a containment hierarchy of hardware entities. Each entity is modeled as a separate row of the ENTITY-MIB's entPhysicalTable.

FIGURE 5-1 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* (see TABLE 5-1).

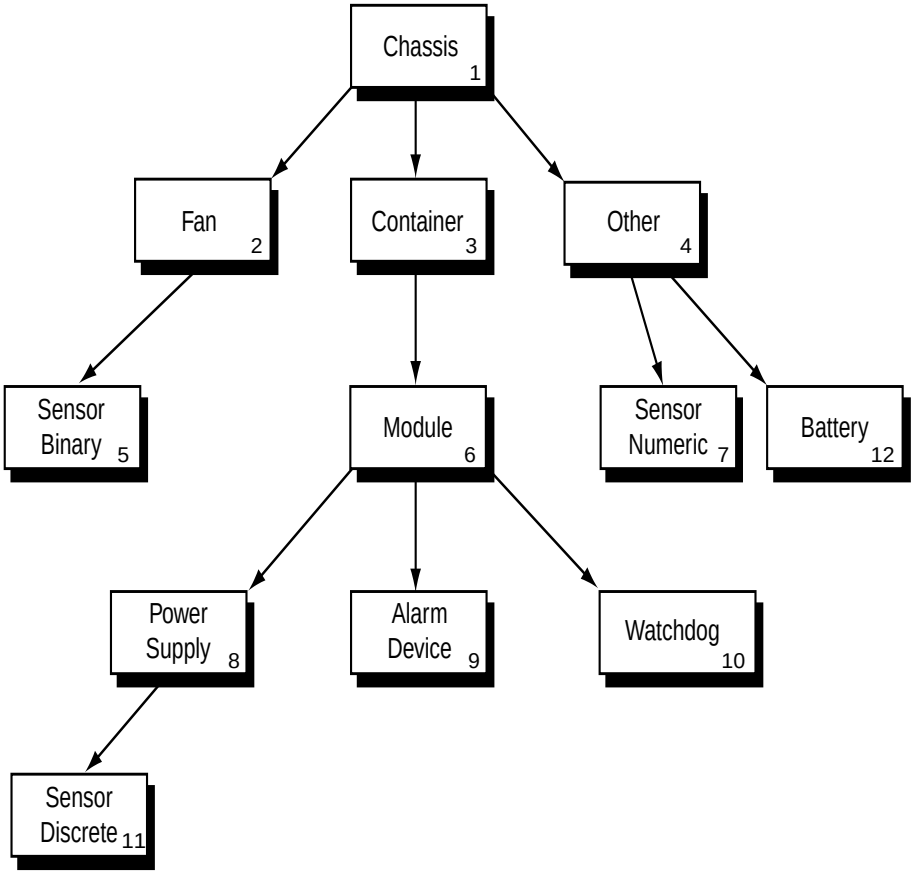


**FIGURE 5-1** Example Hardware Resource Hierarchy

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:

  - The Physical class (*entPhysicalClass*)
  - Common characteristics of the hardware entity
  - A unique index (*entPhysicalIndex*)
  - A 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 firstly by the *entPhysicalIndex* of the containing entry, and secondly by the *entPhysicalIndex* of each contained entry.

TABLE 5-1 shows the *entPhysicalTable* on which the above figure is based, and TABLE 5-2 shows the physical mapping.

**TABLE 5-1** 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 5-2** 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` in 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 enumerations apply to the Sun Fire B1600 platform (see also FIGURE 5-1 and TABLE 5-1):

- `other(1)`

  The enumeration `other` 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.

- `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`

# The Logical Model

The sunPlat logical model uses the ENTITY-MIB to provide a list of high value logical entities. Each entity is represented as a separate row in the ENTITY-MIB's *entLogicalTable*. Note that, unlike the physical model, the logical is flat rather than hierarchical.

The ENTITY-MIB 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 7.

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 ENTITY-MIB 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 ENTITY-MIB 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 8.

# The SUN-PLATFORM-MIB

The SUN-PLATFORM-MIB:

- 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 Fire B1600 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)` entPhysical class.

- 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 5-3 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 5-1

| 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 | | | | watch dog | | | | | | | | | |
| 4 | other | | | | other | | | | | | | | | |
| | | | | | | | | | | | | | | |
| entPhysicalTable | | sunPlatEquipmentTable | sunPlatEquipmentHolderTable | sunPlatCircuitPackTable | sunPlatPhysicalTable | sunPlatWatchdogTable | sunPlatFanTable | sunPlatAlarmTable | sunPlatSensorTable | sunPlatBinarySensorTable | sunPlatNumericSensorTable | sunPlatDiscreteSensorTable | sunPlatDiscreteSensorStatusTable | sunPlatPowerSupplyTable |

**TABLE 5-3** Physical Entity Table Extensions

**TABLE 5-4** Key to Physical Entity Table Extensions (TABLE 5-3)

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

# Logical Model Table Extensions

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 extensions:

- Logical Class Extension Table

  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)`.

    A set of entries in the Load Info Table (*sunPlatInitialLoadInfoTable*) is associated with each Computer System logical entity. This set comprises parameters used to control the boot setting of the computer system.

# Event and Alarm Log Tables

SNMP traps are not guaranteed to be delivered. In view of this, and to support management applications ability to track accurately the current status of platform alarms, the MIB maintains current problem lists for each managed object. This is a table of outstanding alarms for each object. These are cleared automatically when the alarm condition clears.

The SUN-PLATFORM-MIB defines logs that can be used to record events or alarms grouped by event or alarm type, or by affected entity. The implementation for the Sun Fire B1600 employs these logs to maintain lists of outstanding alarms for every monitored entity for the reason stated in the preceding paragraph.

Each entity in the MIB for which alarms can be generated (that is, all physical and logical entities) has an entry in the Log Table (sunPlatLogTable). This table provides administrative status and control for the current problem lists. The logs are permanently enabled and have no limit on size.

There can be zero or more entries in the Log Record Table corresponding to each entry in the Log Table. These entries are described in detail in Chapter 8.

# Event Records

These records form part of the sunPlat trap notifications. Changes in the model are communicated to management applications using two categories of SNMP traps—Events and Alarms.

## Events

- Object Creation Record

  This record indicates that a resource has been added to the object model.

- Object Deletion Record

  This record indicates that a resource has been removed from the object model.

- State Change Record

  This record indicates that the state of the resource has changed.

- Integer Attribute Value Change Record

  This record indicates a change in a characteristic of a resource modeled by an attribute of type `INTEGER`. The integer can be signed or unsigned, depending on the affected object.

- String Attribute Value Change Record

  This record indicates a change in a characteristic of a resource modeled by an attribute of type `OCTET STRING`.

- OID Attribute Value Change Record

  This record indicates a change in the object identifier attribute of type `OBJECT IDENTIFIER`.

## Alarms

- Communications Alarm Record

  This record indicates that a failure has occurred in the communication services that the resource supports.

- Environmental Alarm Record

  This record indicates an environmental condition relating to the resource.

- Equipment Alarm Record

  This record indicates that a resource has become faulty.

- Processing Error Alarm Record

  This record indicates that a resource has an associated software or processing fault.

- Quality of Service Alarm Record

  This record indicates that a quality of service alarm has occurred.

- Indeterminate Alarm Record

  This record indicates that an alarm of unknown type has occurred.

CHAPTER 6

# The 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:



## sunPlat Physical Class Hierarchy

FIGURE 6-1 shows the inheritance hierarchy of the sunPlat classes used to model hardware resources within the Sun Fire B1600.

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.

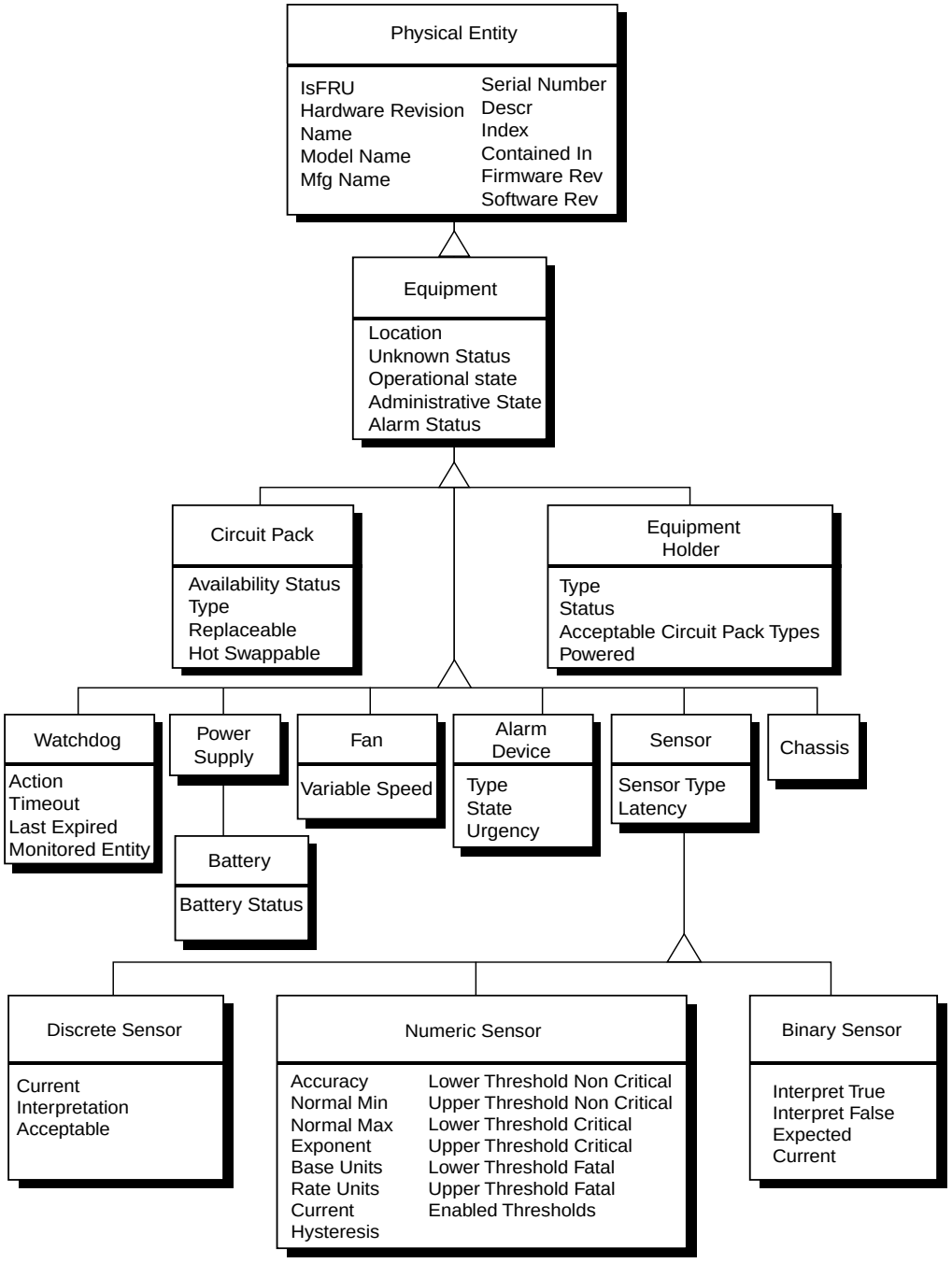


**FIGURE 6-1** The sunPlat Physical Resource Inheritance Class Diagram

# 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 which 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.
- ***Is FRU***
  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.
- ***Hardware Rev***
  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.
- ***Model Name***
  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.
- ***Serial Num***
  This is a text string containing the Manufacturer's serial number for the resource. Not all hardware resources have associated serial numbers.

- ***Mfg Name***
  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 6-1:

**TABLE 6-1** Physical Entity Superclass ‘Class’ Attribute Mapping

| entPhysicalClass | sunPlat Class |
|---|---|
| chassis(3) | *sunPlat Chassis* |
| backplane(4) | Not implemented |
| container(5) | *sunPlat Equipment Holder* |
| powerSupply(6) | *sunPlat Power Supply* |
| fan(7) | *sunPlat Fan* |
| sensor(8) | *sunPlat Sensor*, plus subclasses |
| module(9) | *sunPlat Circuit Pack* |
| port(10) | Not implemented |
| stack(11) | Not implemented |
| other(1) | *sunPlat Equipment*, 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 may vary on each invocation of the agent.
- ***Contained In***
  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`.

- ***Firmware Rev***
  This is a text string containing the manufacturer's firmware revision information for the resource. Not all hardware resources have associated firmware revision information.
- ***Software Rev***
  This is a text string containing the manufacturer's software revision information for the resource. Not all hardware resources have associated software revision information.

# 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 resource.

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.

- ***Administrative State***
  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)`

- ***Operational State***
  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 6-2.

**TABLE 6-2** 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. |

- ***Alarm Status***
  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)`
- ***Unknown Status***
  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`.
- ***Location Name***
  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.

# 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.
- ***Availability Status***
  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 6-3. Not all of these are applicable to every class of managed object. This attribute contributes to the *state* of the managed object.

**TABLE 6-3** 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 6-3** 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.
- ***Hot Swappable***
  This read-only attribute takes a boolean value indicating whether the replaceable resource is hot swappable.

# 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 6-4:

**TABLE 6-4** 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. sunPlat interprets its use within a chassis as a physical receptacle requiring cables for signal connections |
| `shelf(2)` | A horizontal support or sub rack 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. |

- ***Acceptable Types***
  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 6-5.

**TABLE 6-5** 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 *Acceptable Circuit Pack Types* list |
| `notInTheAcceptableList(3)` | The holder contains a removable resource recognizable by the network element; but not one of the types in the *Acceptable Circuit Pack Types* 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)`

# 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)`

# 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)`

# 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 6-6.

**TABLE 6-6** 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 reset 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 |

- ***Last Expired***
  This read-only attribute indicates the date and time at which the watchdog last expired.
- ***Monitored Entity***
  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)`

# 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 entPhysicalClass `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 6-7.

**TABLE 6-7** 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 6-8.

**TABLE 6-8** 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)`

## 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 sunPlat Sensor.

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)`

## 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 6-9.

**TABLE 6-9** Sensor Type Attribute Values

| Type | Description |
|---|---|
| `temperature(3)` | Sensor for measuring the environmental temperature |
| `voltage(4)` | Sensor for measuring the electrical voltage |
| `current(5)` | Sensor for measuring the electrical current |
| `tachometer(6)` | 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.

## 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.
- ***Interpret True***
  This read-only attribute is a text string indicating the interpretation of a `true` value from the sensor.
- ***Interpret False***
  This read-only attribute is a text string indicating the interpretation of a `false` value from the sensor.

## 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 milliampere 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.

---

- ***Base Units***
  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`.
- ***Rate Units***
  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.
- ***Normal Min***
  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.
- ***Normal Max***
  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.
- ***Lower Non Critical Threshold***
  This read-only attribute is an integer indicating the lower threshold at which a `nonCritical` condition occurs.
- ***Upper Non Critical Threshold***
  This read-only attribute is an integer indicating the upper threshold at which a `nonCritical` condition occurs.

- ***Lower Critical Threshold***
  This read-only attribute is an integer indicating the lower threshold at which a `critical` condition occurs.
- ***Upper Critical Threshold***
  This read-only attribute is an integer indicating the upper threshold at which a `critical` condition occurs.
- ***Lower Fatal Threshold***
  This read-only attribute is an integer indicating the lower threshold at which a `fatal` condition occurs.
- ***Upper Fatal Threshold***
  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.
- ***Enabled Thresholds***
  This is read-only attribute that, when written to, resets the sensors to their default values.

## 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 sunPlatDiscreteSensor table has one attribute, *sunPlatDiscreteSensorCurrent*, which indicates the current state of the sensor expressed as an index in the sunPlatDiscreteSensorStates table.

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 sunPlatDiscreteSensorStates table, 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.

## 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 7

# The Logical Model

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

The chapter contains the following sections:



## sunPlat Logical Class Hierarchy

FIGURE 7-1 shows the inheritance hierarchy of the sunPlat logical classes.

The Logical Entity class provides information common to all logical objects.

The Unitary Computer System class adds properties relevant to reporting the power status of the modeled computer systems (for example, a Sun Fire B100s blade in a Sun Fire B1600 chassis), which can also be used to effect forced reset.

The Administrative Domain class adds no additional properties, but is used to represent the logical object representing administrative contact with the modeled system. In the case of the Sun Fire B1600 platform, this is used to represent the System Controller.

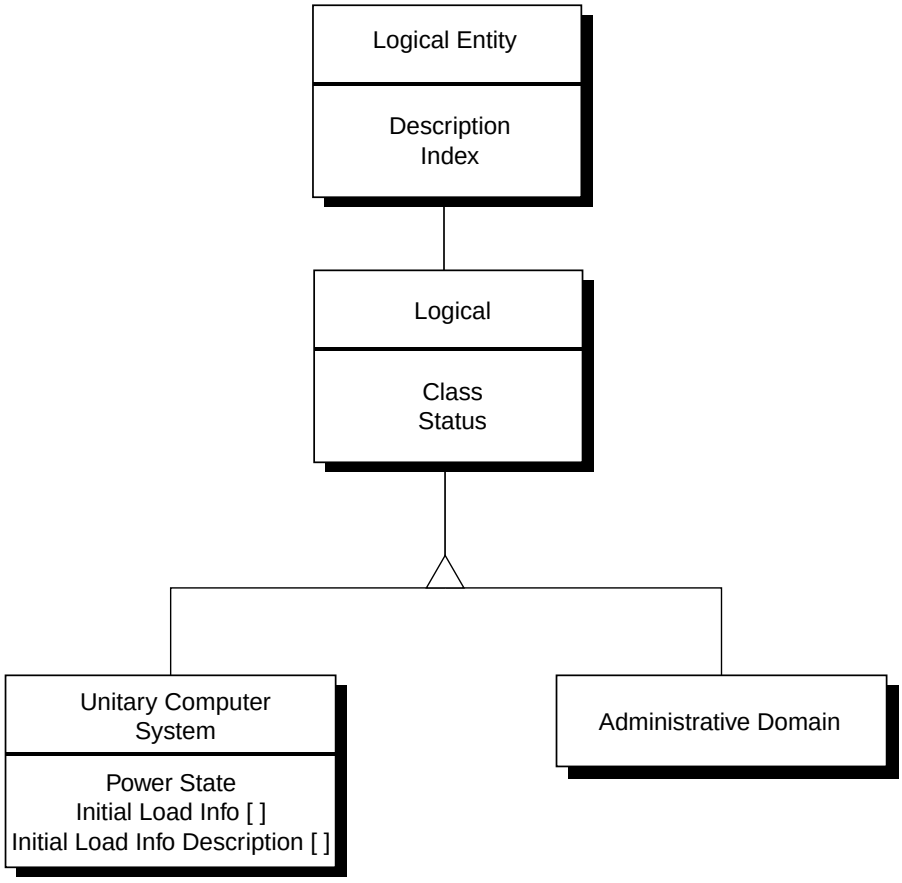


**FIGURE 7-1** The sunPlat Logical Resource Inheritance Class Diagram

# SunPlat Logical Class Definitions

The attributes of the logical sunPlat classes are used to represent the characteristics of logical resources. Such resources represent *high value* objects such as domains in a multi-domain system. 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 objects state.

# Logical Entity

This class represents the logical entity providing identity information. The significant objects are:

---

**Note –** The *entLogical* prefix has been omitted from the following object names for clarity.

---

- ***Description***

  This object identifies the type of object being managed.

- ***TAddress***

  This provides the IP address and UDP port number through which the entity can be managed directly. For a Sun Fire B100s blade in a Sun Fire B1600 system, this gives the IP address of the blade, and port 161, through which the blade's standard Solaris SNMP agent is contacted.

# Logical

This class represents the type of status of the resource represented by this logical entity. The class contains the following objects

---

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

---

- ***Class***

  This attributes in an enumerated type indicating the type of logical class, and takes the following values:

  - `other(1)`
  - `computerSystem(2)`
  - `adminDomain(3)`

- ***Status***

  This attribute is an enumerated type indicating the status of the logical class. It can take the following values:

  - `ok(1)`
  - `error(2)`
  - `degraded(3)`
  - `unknown(4)`

- predFail(5)
- starting(6)
- stopping(7)
- service(8)
- stressed(9)
- nonRecover(10)
- noContact(11)
- lostComm(12)
- stopped(13)

## sunPlat Unitary Computer System

The specific properties of this class are represented using the sunPlatUnitaryComputerSystemTable. It has the sunPlat Logical Class of computerSystem(2)

The class contains the following objects:

---

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

---

- ***Power State***

  This attribute indicates the current power state when read. It also enables remote control of the power state, for example, to power a blade up or down, or to power cycle it to effect a forced reset.

  The attribute can take the following values:

  - unknown(1)
  - fullPower(2)
  - psLowPower(3)
  - psStandby(4)
  - psOther(5)
  - powerCycle(6)
  - powerOff(7)
  - psWarning(8)
  - hibernate(9)
  - softOff(10)
  - reset(11)

- ***Apply Settings***

  Writing to this property enables either the default, or a custom set of boot parameters to be applied.

  Associated with each entry in the sunPlatUnitaryComputerSystemTable is a set of entries in the *sunPlatInitialLoadInfoTable* defining both the current boot parameter setting and an alternate set that can be applied by writing to the *sunPlatUnitaryComputerSystemApplySettings* object.

# sunPlat Administrative Domain

This class adds no properties to the Logical Entity class and thus has no associated MIB objects. The class has the sunPlat Logical Class of `adminDomain(3)`.

CHAPTER 8

# The sunPlat Notifications

This chapter describes the SunPlat notifications classes and attributes, as defined in the SUN-PLATFORM-MIB.

sunPlat notification classes are asynchronous messages sent by the agent to registered network managers. They are used to convey event information more efficiently than can be achieved through polling the managed objects.

The chapter contains the following sections:



## sunPlat Notifications Class Hierarchy

FIGURE 8-1 shows the inheritance hierarchy of the sunPlat Notifications classes.

The set of Notification classes are represented using an hierarchy of both abstract and concrete classes exploiting the common attributes across these classes.

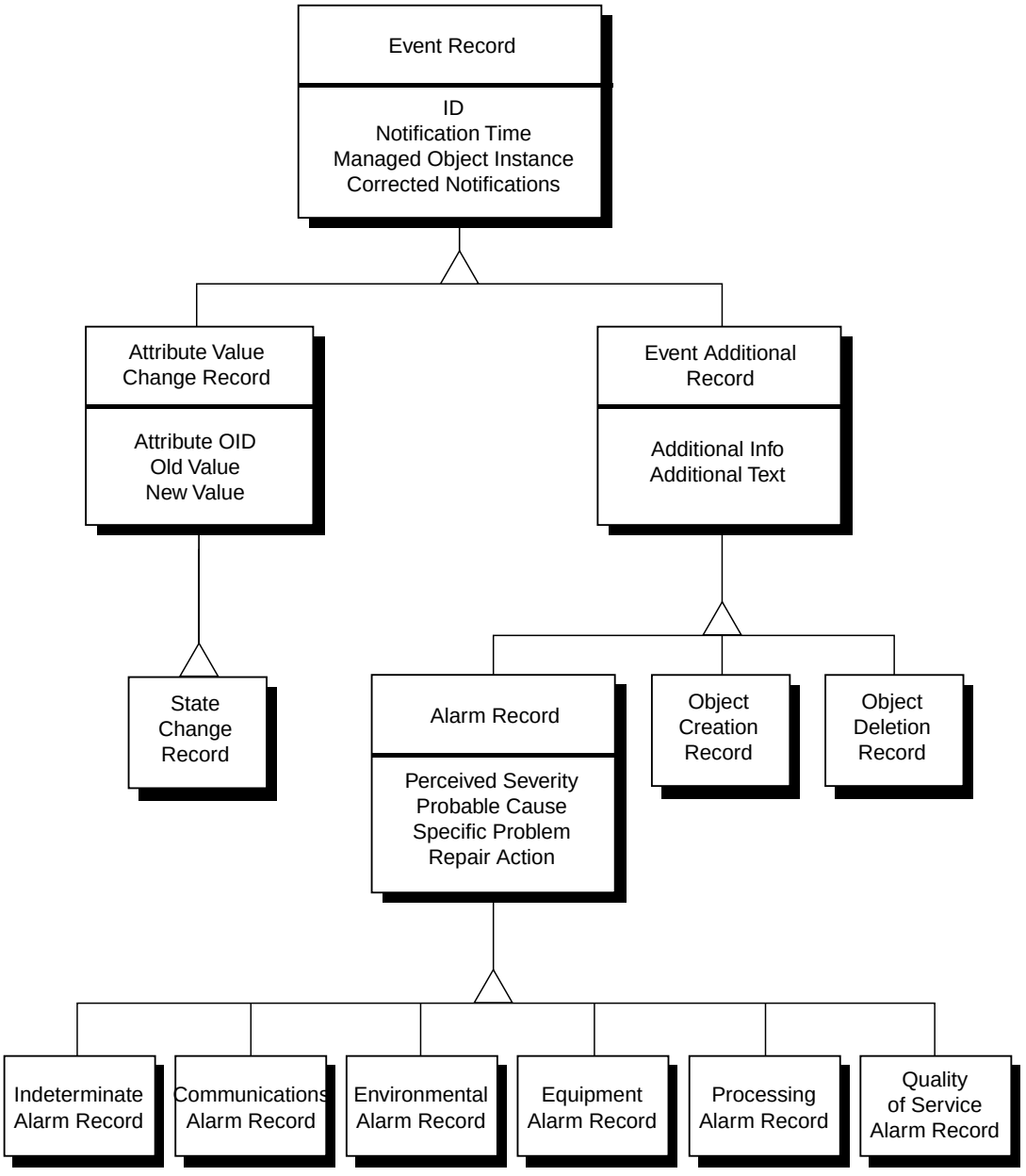


**FIGURE 8-1** Event Records Inheritance Class Diagram

# sunPlat Class Definitions

## sunPlat Event Record

The sunPlat Event Record superclass represents the attributes common to all notifications. This class is further subclassed to provide additional information pertinent to the particular event that it records.

The sunPlat Event Record has the following attributes:

---

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

---

- ***ID***
  This is an integer uniquely identifying the notification and an indication of the order in which the notifications were generated by the agent. Note that the agent does not guarantee that its sequencing reflects the order of the underlying events from which the notifications were generated.
- ***Notification Time***
  This read-only attribute is a timestamp that currently identifies the time at which the notification was generated.
- ***Managed Object Instance***
  This read-only attribute is an OID that provides a direct reference to an entry in the MIB representing the resource with which the event is associated.
- ***Correlated Notifications***
  This read-only attribute is a comma-separated list of ID values that identify the other events to which this event is associated.

## sunPlat Event Additional Record

The sunPlat Event Additional Record superclass represents the additional attributes common to notifications that are generated when the following events occur:

- Object Creation
- Object Deletion
- Alarms

This class is further subclassed to provide additional information applicable to the particular event that it records.

The sunPlat Event Additional Record has the following attributes:

---

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

---

- ***Additional Info***
  This read-only attribute is an optional OID of an object that can provide additional information relevant to this notification.
- ***Additional Text***
  An read-only attribute optional text string that provides additional information relevant to this notification identifying the affected component by its label and `entPhysical` name.

## sunPlat Object Creation Record

The sunPlat Object Creation Record class is used to indicate that a resource has been added to the hierarchy below the associated resource; this may be due to a hot-plugging event. The *Additional Info* attributes contains the OID of the *Physical Entity Table* entry representing the added resource.

Logical objects are created under a manager object instance of `0.0`.

## sunPlat Object Deletion Record

The sunPlat Object Deletion Record is used to indicate that a resource has been removed from the hierarchy below the associated resource. The *Additional Info* attributes contains the OID of the *Physical Entity Table* entry representing the removed resource.

---

**Note –** This OID is no longer valid but can still be of use to the receiving manager

---

# sunPlat Alarm Record

The sunPlat Alarm Record superclass represents the additional attributes common to all notifications representing alarms.

This class is further subclassed for identification of the class of alarm that occurred.

The sunPlat Alarm Record has the following attributes:

---

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

---

- ***Perceived Severity***
  This read-only attribute is an enumerated type defining six severity levels that indicate how the service of the resource has been affected by the problem. These values are shown in TABLE 8-1.

**TABLE 8-1** sunPlat Alarm Record Perceived Severity Values

| Perceived Severity | Description |
|---|---|
| `indeterminate(1)` | The severity level for the alarm can not be determined. |
| `critical(2)` | A service affecting condition has occurred and an immediate corrective action is required. |
| `major(3)` | A service affecting condition has occurred and an urgent corrective action is required. |
| `minor(4)` | A non-service affecting condition has occurred and corrective action should be taken in order to prevent a more serious condition arising. |
| `warning(5)` | A potential or impending service affecting fault condition has been detected and action should be taken to prevent a more serious condition arising. |
| `cleared(6)` | This clears all alarms for this resource of the same alarm class that have the same *Probable Cause* and *Specific Problem* (if given). |

- ***Probable Cause***
  This read-only attribute is an optional enumerated type that provides further qualification as to the type of condition that caused an alarm to be generated. Examples of values of this type are:
  - `coolingSystemFailure(134)`
  - `IODeviceError(75)`
  - `powerProblem(58)`
  - `softwareProgramError(283)`
- ***Specific Problem***
  This read-only attribute is an optional text string that identifies further refinements to the *Probable Cause* of the alarm.
- ***Repair Action***
  This read-only attribute is a string that lists the recommended repair actions.

## sunPlat Indeterminate Alarm Record

The sunPlat Indeterminate Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record any alarm that does not fall into any of the following classes:

## sunPlat Communications Alarm Record

The sunPlat Communications Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record that the associated resource has detected a communications error.

## sunPlat Environmental Alarm Record

The sunPlat Environmental Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record that the associated resource has detected a problem in the environment.

## sunPlat Equipment Alarm Record

The sunPlat Equipment Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record that the associated resource has detected a fault.

## sunPlat Processing Alarm Record

The sunPlat Processing Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record that the associated resource has detected a software or processing failure.

## sunPlat Quality of Service Alarm Record

The sunPlat Quality of Service Alarm Record class does not extend the information provided by the sunPlat Alarm Record class. This class is used to record that the associated resource has detected a change to the quality of service.

# sunPlat Attribute Value Change Record

The sunPlat Attribute Value Change Record superclass represents the additional attributes common to notifications representing attribute changes in the associated resource.

This class is further subclassed for each of the possible attribute types.

The sunPlat Attribute Value Change Record has the following attributes:

---

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

---

- ***OID***
  This read-only attribute is an OID that provides a direct reference to an object in the Physical Entity Table or Logical Entity Table that represents the managed object's attribute whose value has changed.

Depending on the syntax of the affected attribute, the new and old values are represented using one of the following pairs of objects:

- ***New Integer***
  This read-only attribute identifies the new `INTEGER` value of the changed attribute of the managed object. The type signed or unsigned, corresponds to that of the changed attribute.
- ***Old Integer***
  This read-only attribute identifies the old `INTEGER` value of the changed attribute of the managed object. The type, signed or unsigned, corresponds to that of the changed attribute.
- ***New String***
  This read-only attribute identifies the new `OCTET STRING` value in an attribute change notification.
- ***Old String***
  This read-only attribute identifies the old `OCTET STRING` value in an attribute change notification.
- ***New OID***
  This read-only attribute identifies the new `OBJECT IDENTIFIER` value in an attribute change notification.
- ***Old OID***
  This read-only attribute identifies the old `OBJECT IDENTIFIER` value in an attribute change notification.

## sunPlat State Change Record

The sunPlat State Change Record class does not extend the information provided by the sunPlat Attribute Value Change Record class. This class is used to indicate an change in the managed object's attribute that reflects an aspect of the *state* of the resource.

# PART 2 Installation and Configuration

CHAPTER 9

# The Management Software Components

This chapter describes the components that make up the management software for the Sun Fire B1600 and lists the requirement for installing the SNMP software.

The chapter contains the following sections:



## System Management Options

The following system management options are provided for the Sun Fire B1600:

- System monitoring and control using SNMP
- SNMPv3 functionality supporting secure management
- System monitoring using Sun Management Center 3.0[1]

1. For a detailed description, including installation and configuration, refer to the *Sun Management Center 3.0 Supplement for the Sun Fire B1600* (part no. 817-1011-10)

# Instrumentation

Depending on the platform type, you can employ:

- A domain agent, running on the Sun Fire B100s blade (domain hardware monitoring)

  The software is installed locally on the server being monitored and only that server can be monitored. In the case of the Sun Fire B1600, each blade is monitored separately and only one blade can be viewed by each agent instance.

- A platform agent, proxied through a system controller (platform hardware monitoring)

  The software is installed on a remote server that accesses platform instructions by means of the system controller. This enables you to monitor all the hardware managed by the system controller. In the case of the Sun Fire B1600, a whole shelf of blades can be monitored, including all blades of any type, power supplies and system controllers.

In FIGURE 9-1, platform hardware monitoring is employed for Sun Fire B1600 shelves A and B, and domain hardware monitoring is employed for Sun Fire B1600 shelf C.

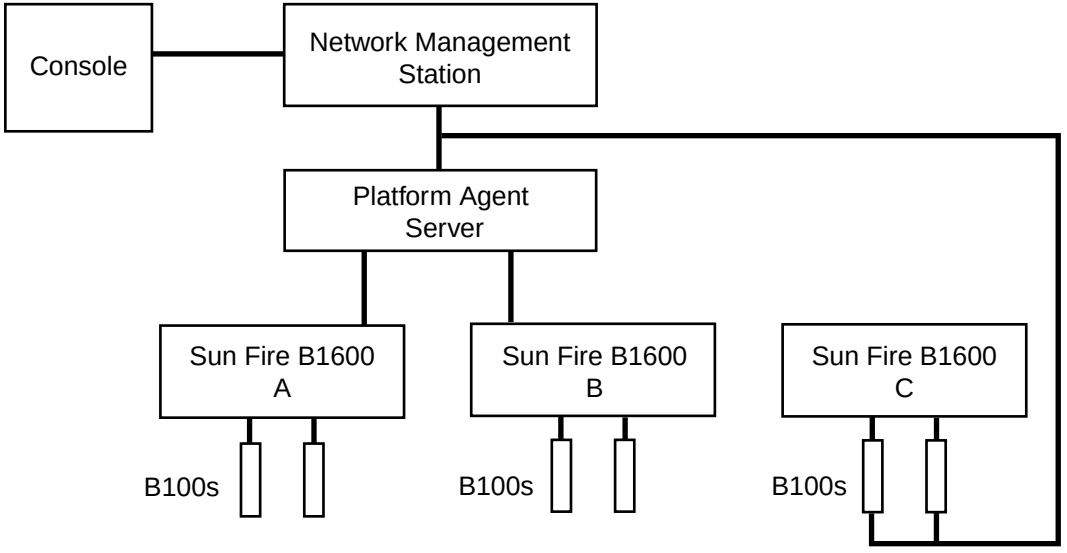


**FIGURE 9-1** Example of Domain and Hardware Platform Monitoring

# System Requirements

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

## Operating Environment

The SNMP Management Agent software requires Solaris 8 Update 3 or later.

## Disk Space Requirements

At least 512 MByte must be available on the platform agent server (1.0 GByte is recommended).

## Patches

The following patches must be installed in addition to the standard Solaris operating environment software:

### Solaris 8

No patches are required for the Sun Fire B100s blade.

Java 1.4 must be installed on the platform agent server and blade (for both platform and domain hardware monitoring) before you install the SNMP Management Agent software (see “Java Environment” on page 70).

### Solaris 9

No patches are required.

# Java Environment

To monitor a Sun Fire B100s blade fully, you must pre-install Java J2SE 1.4 components on each monitored Sun Fire B100s blade and on the platform agent server.

---

**Note –** This installation upgrades any existing J2SE software. If you do not want to upgrade the software—for example, because you have applications that have been qualified against the default J2SE version 1.3.1—you can install the J2RE to co-exist with the default system J2SE. This requires additional configuration of the Sun SNMP Management Agent software, which is described in Appendix A.

---

If you are monitoring only the Sun Fire B1600 shelf without the target instrumentation, you need to pre-install Java J2SE 1.4 components only on the platform agent server. In this case, the instrumentation for the hard disk drive, CPU information and the Ethernet MAC address is not available.

To ensure that the Java 1.4 files are installed in the correct location (`/usr/j2se`), use the `j2sdk-1_4_0_03-solaris-sparc.tar.Z` package to install them.

The file is available from

`http://java.sun.com/j2se/1.4/download.html`

Select the SDK download for `Solaris SPARC 32-bit tar.Z`

Follow the instructions for this download that are available at the above location.

---

**Note –** This filename is correct at the time of writing. Ensure that you have the latest version of this file. The file name has the format `j2sdk-1_4_0_<ver>-solaris-sparc.tar.Z,` where `<ver>` is the revision of the software.

---

As this installation replaces the system J2SE, to ensure any existing Java applications continue to run correctly, you must also install the 64-bit J2SE 1.4 packages, which are contained in the file `j2sdk-1_4_0_<ver>-solaris-sparcv9.tar.Z`.

---

**Caution –** J2SE 1.4 is intended to replace J2SE 1.3.1 on Solaris 8 and you must uninstall the latter before you install J2SE 1.4. If you install a subsequent quarterly update for Solaris 8, some of the J2SE 1.4 packages will be overwritten by J2SE 1.3.1 packages. To ensure that J2SE 1.4 is installed in the correct locations, use `pkgadd` to install it.

---

## Confirming Installation

To make sure you have the correct installation, use the following command:

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-
b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

This reports the version installed on your system.

# Java SNMP API

The installation packages include a newer version of the Java SNMP API, `SUNWjsnmp`. Remove the existing version of this package using `pkgrm` before installing the Management Agent software.

# Installation Packages

The packages that comprise the management software can be divided into the following groups:

- Packages that are required for domain hardware monitoring
- Packages that are required for platform hardware monitoring on the platform agent server
- Packages that are required for platform hardware monitoring on the target machine

These packages are shown in TABLE 9-1.

**TABLE 9-1** SNMP Management Agent Software Package Descriptions

| Package | Package Name | Function |
|---|---|---|
| SUNWbgpc | SPA Personality Module Framework | Framework supporting personality modules |
| SUNWbgptk | SPA Personality Module Toolkit | Reusable library of component models and data access libraries |
| SUNWbgpr | SPA Personality Module (root) | RDP startup script |
| SUNWbgcm | SPA HW Platform Object Manager | Platform object manager |
| SUNWbgcmr | SPA HW Platform Object Manager (root) | Platform object manager startup script |
| SUNWbgpm | SPA SNMP Protocol Mediator / Master Agent | SNMP protocol support and Master Agent |
| SUNWbgpmr | SPA SNMP Protocol Mediator / Master Agent (root) | SNMP component startup script |
| SUNWbgidr | SPA Domain Discovery (root) | Domain Agent Discovery startup script |
| SUNWbgod | SPA Platform Discovery | Platform Agent Discovery daemon |
| SUNWbgodr | SPA Platform Discovery (root) | Platform Agent Discovery startup script |
| SUNWbgpji | SPA Sun Fire B100s Domain Personality Module | B100s domain instrumentation |
| SUNWbgpjo | SPA Sun Fire B1600 Platform Personality Module | B1600 platform instrumentation |

**Note –** Internal dependencies exist between the packages and they must be installed in a specific order (see “Installing the SNMP Software” on page 79.

## Upgrading the Software

To upgrade the software, you must remove the existing software before reinstalling the new version (see Chapter 13).

# Package Delivery

The packages are supplied in tar archive bundles. TABLE 9-2 shows the contents of the `SUNWspa.1.0.tar.Z` archive bundle, which contains the packages that provide SNMP support for platform management of a Sun Fire B1600 and domain management of a Sun Fire B100s.

When unpacked, the packages are located in the directories shown in the table. Refer to chapter 10 for detailed instructions for installing the software.

**Note –** Ensure that you have the latest version of this file.

**TABLE 9-2** SNMP Management Agent Package Bundle

| Bundle | Description | Contents |
|---|---|---|
| SUNspa.1.0.22.tar.Z | SNMP platform agent packages to be installed on the platform agent server | platform/proxy/SUNWbgpc<br>platform/proxy/SUNWbgpk<br>platform/proxy/SUNWbgcm<br>platform/proxy/SUNWbgcmr<br>platform/proxy/SUNWbgod<br>platform/proxy/SUNWbgodr<br>platform/proxy/SUNWbgpjo<br>platform/proxy/SUNWbgpm<br>platform/proxy/SUNWbgpmr<br>platform/proxy/SUNWjdrt<br>platform/proxy/SUNWjsnmp |
| | SNMP instrumentation packages to be installed on the Sun Fire B100s blade | platform/target/SUNWbgpc<br>platform/target/SUNWbgptk<br>platform/target/SUNWbgpr<br>platform/target/SUNWbgcm |
| | SNMP domain agent packages to be installed on the Sun Fire B100s blade | domain/SUNWbgpc<br>domain/SUNWbgptk<br>domain/SUNWbgcm<br>domain/SUNWbgcmr<br>domain/SUNWbgidr<br>domain/SUNWbgpji<br>domain/SUNWbgpm<br>domain/SUNWbgpmr<br>domain/SUNWjdrt<br>domain/SUNWjsnmp |

To unpack the tar file, type:

```
$ zcat SUNWspa.1.0.tar.Z | tar xf -
```

## Installing the Domain or Target Packages on the Sun Fire B100s

The domain (in the case of domain hardware monitoring) or target (in the case of platform hardware monitoring) packages must be installed on the Sun Fire B100s blades. There are several ways you can achieve this, including the following:

- Unpacking the domain or target packages on each individual Sun Fire B100s blade
- Unpacking the domain or target packages in a shared directory visible to root user of each Sun Fire B100s blade on which the software is to be installed
- Setting up the domain or target packages for network installation

# Effect on System Files

Several new startup files are created in `/etc/init.d`, as shown in TABLE 9-3, with links to `/etc/rc<n>.d`.

**TABLE 9-3** Startup Scripts

| Component | Startup Script | Package Name | Package Description |
|---|---|---|---|
| Platform Object Manager | `spapom` | `SUNWbgcmr` | Platform Object Manager (root) |
| Domain Hardware Discovery Module | `spaibdm`* | `SUNWbgidr` | Domain Hardware Discovery Module (root) |
| Remote Data Plug-ins | `spardp`† | `SUNWbgpr` | Personality Module (root) |
| SNMP Protocol Mediator / Master Agent | `spama` | `SUNWbgpmr` | SNMP Protocol Mediator SNMP Master Agent (root) |

* On the Sun Fire B100s blade only with domain packages.

† On the Sun Fire B100s blade only with platform / target packages.

The Discovery Module is started automatically by `inetd` and a new entry is thereby created in the `/etc/inetd.conf` file.

The Platform Object Manager (POM) monitors the activity on an IP port for requests from its clients. These ports are registered in the `/etc/services` file.

CHAPTER 10

# Installation

This chapter describes how to install the management software on the Sun Fire B1600.

The chapter contains the following sections:



## Selecting the Installation

There are two main considerations to be made when you are deciding which configuration of the software to install:

1. Instrumentation configuration
2. Management interface configuration

### Instrumentation Configuration

Depending on the platform type, you can employ:

- A domain agent, running on the monitored system

  Software is installed locally on the Sun Fire B100s blade being monitored (domain). Each blade is monitored separately and only one blade can be viewed at a time. This is known as *domain hardware monitoring*.

- A platform agent, proxied through a system controller

  Software is installed on a remote (platform agent) server, which accesses platform instructions by means of the Sun Fire B1600 system controller, and on the Sun Fire B100s (target) blades to be monitored. This enables you to monitor all the hardware managed by the system controller, including power supplies, system controllers and all the blades. This is known as *platform hardware monitoring*.

  If you are using platform hardware monitoring, to obtain information about hard disk drives, CPUs and Ethernet Mac Addresses, you need to install target packages on each monitored Sun Fire B100s blade.

---

**Note –** The Sun Fire B100s blade is referred to as the *domain* in domain hardware monitoring and the *target* in platform hardware monitoring.

---

See also “Instrumentation” on page 68.

# Management Interface Configuration

The management software is designed to be deployed in a number of ways and currently, the following are supported:

- SNMP using `snmpdx`, without Master Agent (this is the default installation)
- SNMP using the Master Agent and `snmpdx`

  In this case, you must configure the installation manually as described in Chapter 11 and Chapter 12.

# Installing the SNMP Software

This section summarizes the procedure for installing the monitoring software. The detailed process for each type of installation follows in subsequent sections in this chapter.

Before installing the software, ensure that:

- You have the requisite level of Solaris installed on both the domain or target (Sun Fire B100s) and the platform agent server (see “Operating Environment” on page 69).
- You have installed all the necessary patches (see “Patches” on page 69) and any additional essential packages not supplied as part of the SNMP software (see “Java Environment” on page 70).
- You have installed Java 1.4, either by upgrading the existing J2SE or by installing a separate J2RE as described in “Java Environment” on page 70.

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

You must now decide whether you are using domain hardware monitoring or platform hardware monitoring.

- If you select domain hardware monitoring, follow the procedure described in “Installing Software for Domain Hardware Monitoring” on page 79.
- If you select platform hardware monitoring, follow the procedure described in “Installing Software for Platform Hardware Monitoring” on page 81.

## Installing Software for Domain Hardware Monitoring

Install these packages on each blade to be monitored (see “Installing the Domain or Target Packages on the Sun Fire B100s” on page 75).

### ▼ To Install the Software

1. **Make sure that you have installed Java 1.4.**

   See “Java Environment” on page 70.

2. **Make sure that any existing version of `SUNWjsnmp` is removed.**

   See “Java SNMP API” on page 71.

3. **Install the domain agent packages on the Sun Fire B100s blade(s) in the order shown to avoid dependency issues being reported:**

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgidr \
SUNWbgpji SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. **Configure the Java environment.**

   a. **If you have installed J2SE 1.4 as described in "Java Environment" on page 70, ignore this step.**

   b. **If you have installed J2RE 1.4 as described in "Installing J2RE 1.4" on page 125, edit the domain hardware monitoring startup scripts as described in "Domain Hardware Monitoring" on page 127.**

5. **Configure the software.**

   See Chapter 11.

6. **Reboot the Sun Fire B100s blade(s).**

   You have now completed installing the software for domain hardware monitoring. Continue with "Interface Options" on page 86.

7. **Make sure that the processes have started correctly by typing:**

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?       44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
```

   If the output is similar to the above, the processes are running.

# Installing Software for Platform Hardware Monitoring

- Decide whether you are going to install Java 1.4 on the Sun Fire B100s blades (see the discussion in "Java Environment" on page 70). Installation of Java 1.4 is essential if you want to support target instrumentation.
  - To install the software with the target instrumentation, start the installation process at "To Install the Software with Target Instrumentation" on page 81.
  - To install the software without the target instrumentation, start the installation process at "To Install the Software Without Target Instrumentation" on page 83.
- Install the platform agent packages on the platform agent server.
- Install the target platform agent packages on each blade to be monitored, if required.
- Set up the system controller SMS IP address.

## ▼ To Install the Software with Target Instrumentation

1. **Make sure that you have installed Java 1.4 on the server acting as the platform agent.**

   See "Java Environment" on page 70 and Step 4 below.

2. **Make sure that any existing version of `SUNWjsnmp` is removed from the platform agent server.**

   See "Java SNMP API" on page 71.

3. **Install the platform agent packages on the platform agent server in the order shown to avoid dependency problems:**

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgod \
SUNWbgodr SUNWbgpjo SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. **Configure the Java environment.**

   a. **If you have installed J2SE 1.4 as described in "Java Environment" on page 70, ignore this step.**

   b. **If you have installed J2RE 1.4 as described in "Installing J2RE 1.4" on page 125, edit the platform hardware monitoring startup scripts as described in "Platform Hardware Monitoring" on page 128.**

5. **Configure the software.**

   See Chapter 11.

6. **Start the platform agent manually by typing:**

```
# /etc/init.d/spapom start
# /etc/init.d/init.snmpdx stop
# /etc/init.d/spama stop
# /etc/init.d/snmpdx start
# pkill -1 inetd
```

or by rebooting the platform agent server.

7. **Make sure that the processes have started correctly by typing:**

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?       44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
# netstat -a | grep mismi
     *.mismi            *.*             0     0 24576     0 LISTEN
      *.mismi                            *.*  0
0 24576      0 LISTEN
#
```

If the output is similar to the above, the processes are running.

8. **Make sure that you have installed Java 1.4 on the target Sun Fire B100s blade(s) being monitored.**

See "Java Environment" on page 70 and Step 11 below.

9. **Make sure that any existing version of** `SUNWjsnmp` **is removed from the target blade(s).**

See "Java SNMP API" on page 71.

10. **Install the target platform agent packages on the target blade(s).**

These packages enable access to instrumentation data using the monitored machine's Solaris interfaces.

Install these packages in the order shown to avoid dependency problems.

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgpr
```

11. **Configure the Java environment.**

    a. **If you have installed J2SE 1.4 as described in “Java Environment” on page 70, ignore this step.**

    b. **If you have installed J2RE 1.4 as described in “Installing J2RE 1.4” on page 125, edit the target hardware monitoring startup scripts as described in “Platform Hardware Monitoring” on page 128.**

12. **Start the target instrumentation manually by typing:**

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

    or by rebooting the systems.

13. **Make sure that the process has started correctly by typing:**

```
# netstat -an | grep 1099
      *.1099        *.*         0      0 24576      0 LISTEN
```

    If the output is similar to the above, the process is running.

14. **Set the SMS IP address using** `setupsc`**.**

    Continue from “Configuring the System Controller” on page 85.

## ▼ To Install the Software Without Target Instrumentation

1. **Make sure that you have installed Java 1.4 on the server acting as the platform agent.**

   See “Java Environment” on page 70.

2. **Make sure that any existing version of** `SUNWjsnmp` **is removed from the platform agent server.**

   See “Java SNMP API” on page 71.

3. **Install the platform agent packages on the platform agent server in the order shown to avoid dependency problems:**

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgod \
SUNWbgodr SUNWbgpjo SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. **Configure the Java environment.**

**a. If you have installed J2SE 1.4 as described in "Java Environment" on page 70, ignore this step.**

**b. If you have installed J2RE 1.4 as described in "Installing J2RE 1.4" on page 125, edit the platform hardware monitoring startup scripts as described in "Platform Hardware Monitoring" on page 128.**

**5. Configure the software.**

See Chapter 11.

**6. Start the platform agent manually by typing:**

```
# /etc/init.d/spapom start
# /etc/init.d/init.snmpdx stop
# /etc/init.d/spama stop
# /etc/init.snmpdx start
# pkill -1 inetd
```

or by rebooting the platform agent server.

**7. Make sure that the processes have started correctly by typing:**

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?       44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
# netstat -a | grep mismi
     *.mismi            *.*             0     0 24576     0 LISTEN
      *.mismi                            *.*  0
0 24576      0 LISTEN
#
```

If the output is similar to the above, the processes are running.

**8. Set the SMS IP address using** `setupsc`**.**

Continue below.

# Configuring the System Controller

After you have installed the SNMP software, you must set the SMS IP address on the system controller to that of the platform agent server. To do this, log onto the system controller's console, run `setupsc` and add the IP address of the platform agent server.

In the example below, the IP address is set to `10.5.1.1`.

Press [ENTER] after each question to accept the current value until the following line is displayed:

```
Enter the SMS IP address
```

Enter the IP address and press [ENTER], then continue pressing [ENTER] in response to the remaining questions.

---

**Note –** The `setupsc` command is described in the *Sun Fire B1600 Blade System Chassis Software Setup Guide*.

---

**CODE EXAMPLE 10-1** Setting the SMS IP Address

```
hornet-sc>setupsc
Entering Interactive setup mode.
Use Ctrl-z to exit & save. Use Ctrl-c to abort.
Do you want to configure the enabled interfaces [y]?
Should the SC network interface be enabled [y]?
Should the SC telnet interface be enabled for new connections [y]?
Do you want to configure the network interface [y]?
Should the SC use DHCP to obtain its network configuration [n]?
Enter the SC IP address [129.156.174.140]:
Enter the SC IP netmask [255.255.255.0]:
Enter the SC IP gateway [129.156.174.1]:
Do you want to configure the SC private addresses [y]?
Enter the SSC0/SC IP private address [129.156.174.118]:
Enter the SSC1/SC IP private address [129.156.174.128]:
Do you want to enable a VLAN for the SC [n]?
Enter the SMS IP address [0.0.0.0]: 10.5.1.1
<truncated>

hornet-sc>
```

# Interface Options

The default installation provides management through SNMP acting as a sub-agent of `snmpdx`. No user input is required during installation, although you can customize the deployment after configuration.

By editing the configuration files, you can add Master Agent functionality to SNMP and `snmpdx`.

---

**Note –** In all cases, the SNMP Access Control Lists (ACLs) have a default configuration that prevents access. You must configure these to enable access (see Chapter 11).

---

## SNMP using `snmpdx` (Default)

This option registers the SNMP Mediator as a sub-agent of `snmpdx`, using an automatically allocated UDP port number to which requests to the Mediator are directed. These requests can be either through `snmpdx`, or direct, as shown by the dotted line in FIGURE 10-1, if this is enabled in the Mediator ACL file (see Chapter 12).

**TABLE 10-1** Port Summary for FIGURE 10-1

| Key | Function | Parameter in `spama.conf` | Default |
|---|---|---|---|
| 1 | Port to which `snmpdx` forwards requests to the SNMP Mediator | `SPAPM_REQ_PORT` | |
| 2 | Port to which the Mediator sends traps to the SNMP managers | `SPAPM_TRAP_PORT` | 162 |

---

**Note –** Although default configuration is automatic, you must still configure the ACL files for `snmpdx` and/or the Mediator to support your manager configuration.

---

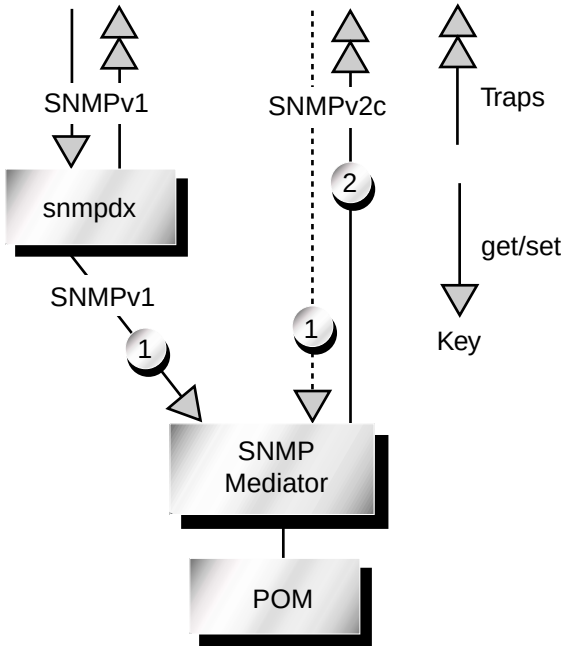


**FIGURE 10-1** Data Flow when SNMP is a Sub-Agent of `snmpdx`

# SNMP Plus Master Agent and snmpdx

This option adds Master AgentSNMPv3 security functionality to SNMP and snmpdx. The snmpdx automatic startup is disabled and the Master Agent is registered on port 161. A new port number is assigned automatically to snmpdx.

Traps from the Mediator are optionally translated by the Master Agent into SNMPv3 or sent directly.

Traps from snmpdx are only forwarded directly to SNMP managers and cannot be translated by the Master Agent.

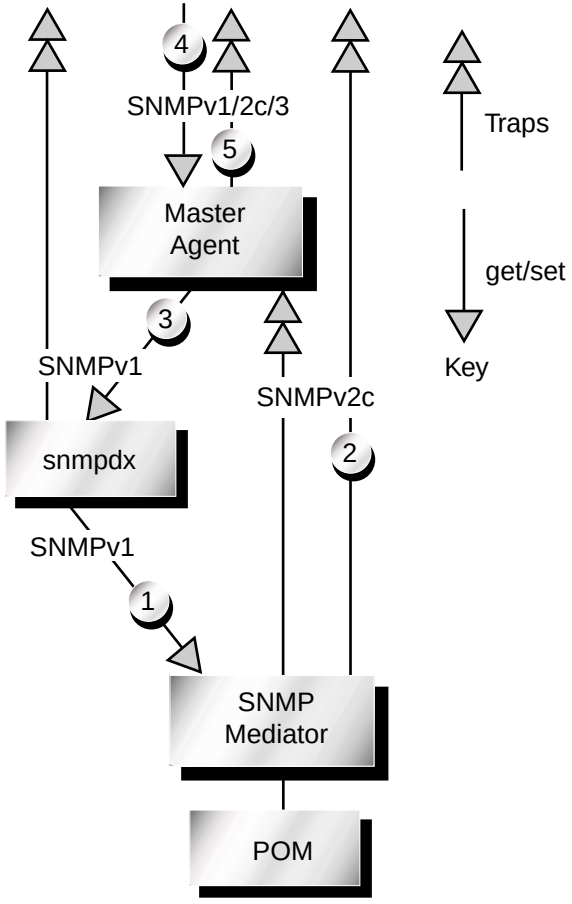


**FIGURE 10-2** Data Flow When Master Agent is Employed

**TABLE 10-2** Port Summary for FIGURE 10-2

| Key | Function | Parameter in `spama.conf` | Default |
|---|---|---|---|
| 1 | Port to which `snmpdx` forwards requests to the SNMP Mediator sub-agent | `SPAPM_REQ_PORT` | |
| 2 | Port to which the Mediator sends traps direct to the SNMP managers | `SPAPM_TRAP_PORT` | 162 |
| 3 | Port to which the Master Agent forwards requests to `snmpdx` | `SNMPDX_REQ_FORWARD_PORT` | |
| 4 | Port monitored by the Master Agent for requests | `MASTER_AGENT_REQ_PORT` | 161 |
| 5 | Port to which the Master Agent sends traps to the SNMP managers | `SPAPM_TRAP_PORT` | 162 |

# Third-party Master Agent Plus SNMP

This option registers the SNMP Mediator as a sub-agent using a port number allocated manually or by the third-party master agent. To enable direct access, you must manually configure the Mediator ACL file as described in Chapter 12.

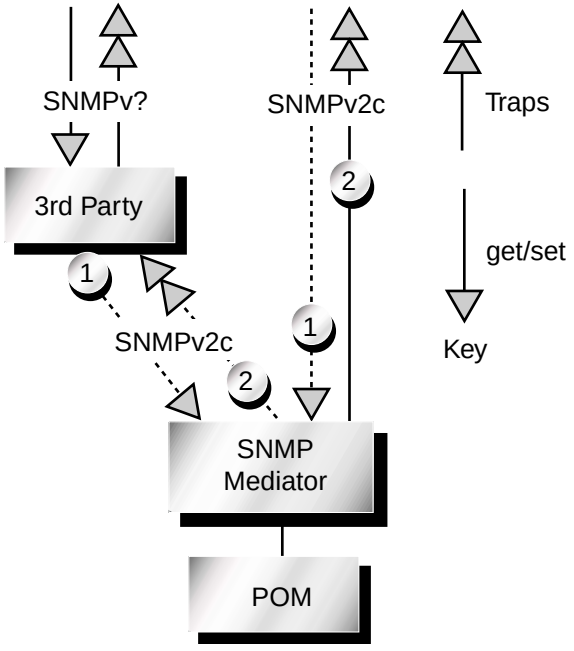


**FIGURE 10-3** Data Flow When a Third-Party Master Agent is Employed

**TABLE 10-3** Port Summary for FIGURE 10-3

| Key | Function | Parameter in `spama.conf` | Default |
|---|---|---|---|
| 1 | Port used by the SNMP Mediator enabling direct access or to which the third-party master agent forwards requests to the SNMP Mediator | `SPAPM_REQ_PORT` | |
| 2 | Port used by the SNMP Mediator to send SNMPv2c traps to the third-party master agent or directly to the SNMP managers | `SPAPM_TRAP_PORT` | |

CHAPTER 11

# Configuration Files

This chapter provides an overview of the files you can edit to configure the software. It lists the configurable parameters and introduces the concept of Access Control.

Read this chapter before referring to Chapter 12, which explains how to configure the basic SNMP options using the files described here.

This chapter contains the following sections:



**Note –** For information about the SNMP packages and how to install them, refer to Chapter 9 and Chapter 10.

# Configuration Files

The following files, which are located in `/etc/opt/SUNWspa/`, determine the configuration of SNMP:

- **General Configuration File**
  - `spama.conf`—Defines how the Master Agent and the Mediator are configured
- **Mediator Configuration Files**
  - `spapm.acl`—Defines the access control for the Mediator
  - `spapm_snmpdx.acl`—Defines the access control for the Mediator as a sub-agent of `snmpdx`
- **Master Agent Configuration Files**

  If you are not using the Master Agent, these is no need to configure these files.
  - `spama.acl`—Defines the access control for the Master Agent
  - `spama.uacl`—Defines SNMPv3 user and context access control for the Master Agent
  - `spama.security`—Defines the SNMPv3 users referenced in `spama.uacl`

These files are described in more detail in the following sections.

# General Configuration File

## `spama.conf`

The `spama.conf` file contains a number of configurable parameters, which are described in the following section and TABLE 11-1. An example of a `spama.conf` file is shown in CODE EXAMPLE 11-1.

## General Options

### START_MEDIATOR

Set this parameter to `yes` to run the Mediator, otherwise set to `no` (see also FIGURE 10-1).

The default value is:

```
START_MEDIATOR=yes
```

### START_MASTER_AGENT

If you want to use SNMPv3 security and you are not using a third-party master agent to provide it, set this parameter to `yes` to start the Master Agent (see FIGURE 10-2 and the accompanying table).

If you do not require SNMPv3 security and you are using another master agent (including snmpdx), or you do not intend to use any master agent, set this parameter to `no` (see also FIGURE 10-1 and FIGURE 10-3 and the accompanying tables).

The default value is:

```
START_MASTER_AGENT=no
```

### AGENT_INTERFACE_NAME

If the Master Agent is enabled (see above), the value set here specifies a network interface to which the Master Agent should bind, the protocol Mediator being bound to `localhost`.

If the Master Agent is not enabled, the value set here defines the host name of the network interface to which the Mediator binds. If you do not specify a value, the default is for access to be by means of the default interface.

If you are using the Mediator as a sub-agent of, for example, `snmpdx`, set the value to `localhost`.

The default value is:

```
AGENT_INTERFACE_NAME=localhost
```

## Master Agent Options

So that the Master Agent can work with snmpdx, the Master Agent startup script stops snmpdx, takes over its SNMP port and restarts snmpdx as a sub-agent on another port.

If the SNMPDX_REQ_FORWARD_PORT parameter has a null value, the Master Agent startup script searches for a free port in the ephemeral anonymous range 32768 through 65535 and restarts snmpdx as a sub-agent on that port. The startup script also searches /etc/services and does not use any of the ports listed there.

However, if you specify the SNMPDX_REQ_FORWARD_PORT value, the Master Agent uses this port to forward requests to snmpdx in which case the Master Agent does not check if the port is already in use.

### MASTER_AGENT_REQ_PORT

This is the port on which the Master Agent receives requests from managers. For most configurations, it is not necessary to change the value (see also FIGURE 10-2 and the accompanying table).

The default value, if unspecified, is 161.

### ENABLE_SNMPV2C_SETS

This parameter controls whether the Master Agent allows set operations to be conducted using SNMPv1 or SNMPv2c. Setting the value to yes reduces security substantially as the SNMPv1 and SNMPv2C protocols are intrinsically insecure.

The default value is:

```
ENABLE_SNMPV2C_SETS=no
```

### SNMPDX_REQ_FORWARD_PORT

This parameter controls the port on which the Master Agent forwards requests to snmpdx. If you do not specify a value, automatic configuration is performed by the Master Agent (see the introduction to this section, and FIGURE 10-2 and the accompanying table).

If you specify a value, you must also configure snmpdx manually to listen on this port.

The default value is

```
SNMPDX_REQ_FORWARD_PORT=
```

### SNMPV3_USER

This parameter determines which SNMPv3 user issues SNMPv3 traps.

The default value is:

`SNMPV3_USER=defaultUser`

---

**Note –** To send SNMPv3 traps, you must set `SPAPM_TRAPS_ARE_V3=yes`.

---

## Protocol Mediator Options

### SUB_AGENT

This parameter determines whether the Mediator or Master Agent is started by a master agent such as snmpdx, instead of automatically on start up.

The default value is:

`SUB_AGENT=yes`

If you specify yes, you must start the Mediator by passing a <port> parameter as follows:

```
# /etc/init.d/spama start <port>
```

where <port> is the UDP port on which the Mediator will listen for SNMP requests. For example, when used with snmpdx, the invocation of the Mediator is controlled by the following line in the `/etc/snmp/conf/spapm.rsrc` file:

```
command = “etc/init.d/spama start $PORT”
```

---

**Note –** If `SUB-AGENT=YES`, the value of `START_MASTER_AGENT` is ignored. If you enable Master Agent functionality, you must set `SUB_AGENT` to `no`.

---

If you specify `no`, the Mediator is launched on startup (by the startup script `/etc/rc3.d/S80spama`), and the UDP port on which it listens for SNMP requests is defined by the `SPAPM_REQ_PORT` setting described below.

### SPAPM_REQ_PORT

This parameter determines the port on which the Mediator receives requests when SUB_AGENT is set to no. See FIGURE 10-1, FIGURE 10-2 and FIGURE 10-3 and the accompanying tables.

The default value is:

```
SPAPM_REQ_PORT=
```

With the default setting, if START_MASTER_AGENT=yes, a port number is automatically allocated. If START_MASTER_AGENT=no, the default port number 33000 is used.

It is intended that if START_MASTER_AGENT=no, you will set SPAPM_REQ_PORT to the required value so that the Mediator can be accessed either by a local master agent using a fixed port, or directly by remote SNMP managers.

### SPAPM_TRAPS_ARE_V3

This parameter determines whether the Mediator traps are SNMPv3 or SNMPv2c.

The default value, which sets the traps to SNMPv2c, is:

```
SPAPM_TRAPS_ARE_V3=no
```

---

**Note –** If you enable SNMPv3 traps (SPAPM_TRAPS_ARE_V3=yes), you must also set START_MASTER_AGENT=yes and START_MEDIATOR=yes.

---

### SPAPM_TRAP_PORT

This parameter determines the port number to which Mediator traps are sent. See FIGURE 10-1, FIGURE 10-2 and FIGURE 10-3 and the accompanying tables.

The default value is:

```
SPAPM_TRAP_PORT=162
```

### SPAPM_TRAP_INTERFACE

This parameter determines the interface from which Mediator sends SNMP traps. If this is not defined, traps are issued from the host's default interface.

**Note –** SNMPv2c traps are sent direct rather than through `snmpdx`.

The default value is undefined:

```
SPAPM_TRAP_INTERFACE=
```

## SPAPM_OPTIONS

This parameter enables you to modify the behavior of the Mediator by specifying one or more of the following options:

- `-a` Send attribute change notifications
- `-s` Send state change notifications
- `-c` Send object creation notifications
- `-C` Enable object creation notifications during initialization
- `-d` Send object deletion notifications
- `-l` Enable current problem list logs by default

The format and default value for this parameter are:

```
SPAPM_OPTIONS="-ascCdl"
```

**TABLE 11-1** Default Values in `spama.conf`

| Parameter | Value following Default Installation |
|---|---|
| START_MEDIATOR | START_MEDIATOR=yes |
| START_MASTER_AGENT | START_MASTER_AGENT=no |
| AGENT_INTERFACE_NAME | AGENT_INTERFACE_NAME=localhost |
| MASTER_AGENT_REQ_PORT | MASTER_AGENT_REQ_PORT=161 (also defaults to this value if unspecified) |
| ENABLE_SNMPV2C_SETS | ENABLE_SNMPV2C_SETS=no |
| SNMPDX_REQ_FORWARD_PORT | SNMPDX_REQ_FORWARD_PORT= |
| SNMPV3_USER | SNMPV3_USER=defaultUser |
| SUB_AGENT | SUB_AGENT=yes |
| SPAPM_REQ_PORT | SPAPM_REQ_PORT= |
| SPAPM_TRAPS_ARE_V3 | SPAPM_TRAPS_ARE_V3=no |

**TABLE 11-1** Default Values in `spama.conf` *(Continued)*

| Parameter | Value following Default Installation |
|---|---|
| SPAPM_TRAP_PORT | SPAPM_TRAP_PORT=162 |
| SPAPM_TRAP_INTERFACE | SPAPM_TRAP_INTERFACE= |
| SPAPM_OPTIONS | SPAPM_OPTIONS=”-ascCdl” |

**CODE EXAMPLE 11-1** Example of a `spama.conf` File

```
#!/sbin/sh
#
#ident  "@(#)spama.conf1.17 01/29/03 SMI"
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.

#
# This file is used to control the configuration of the Master Agent and
# Protocol Mediator
#

#
# Master Agent / Mediator configuration
#

##############################################################################
# General options
##############################################################################

#
# Set to "yes" if the mediator component should be started
#
START_MEDIATOR=yes

#
# Set to "yes" to enable the master agent
#
START_MASTER_AGENT=no

#
# Hostname of the network interface for the agent to bind to. If this
# is not specified the agent will be accessible via the default
# interface.
#
# If the mediator is being used as a sub-agent this should be
# set to localhost.
#
```

**CODE EXAMPLE 11-1** Example of a `spama.conf` File *(Continued)*

```
# If the master agent is enabled, this setting applies to its interface,
# the protocol mediator being bound to localhost.
AGENT_INTERFACE_NAME=localhost

###############################################################################
# Master Agent options
###############################################################################

#
# SNMP port for master agent to receive SNMP get/set requests.
#
# This port number will be used to listen for SNMP get/set
# requests.
#
# If this value is blank, default will be 161 if START_MASTER_AGENT=yes
#
MASTER_AGENT_REQ_PORT=

#
# set to "yes" to enable SNMPv1/SNMPv2c SET operations via the master agent.
#
ENABLE_SNMPV2C_SETS=no

#
# SNMP sub-agent port to which non-SNMP Protocol Mediator (snmpdx) requests
# will be sent.
#
# If this port setting is blank (default), automatic configuration will be
# performed. The port number for snmpdx will be dynamically determined
# if snmpdx is already using the UDP port where the Master Agent listens
# for SNMP get/set requests (by default UDP port 161) at Master Agent startup.
#
# If this port setting is blank but snmpdx is not using the same port
# as the Master Agent will listen for SNMP get/set requests, the Master
# Agent will use the port number which is being used by snmpdx to forward
# the SNMP set/get requests to snmpdx.
#
# If this port is set, it is expected that the user should perform
# the "listening" port configuration for the sub-agents and the Master
# Agent will use this port number to forward non-SNMP Protocol Mediator
# requests.
#
SNMPDX_REQ_FORWARD_PORT=

#
# SNMPv3 user
#
```

**CODE EXAMPLE 11-1** Example of a `spama.conf` File *(Continued)*

```
# The Mediator will use this user to send the V3 traps (if enabled with
# SPAPM_TRAPS_ARE_V3).
#
# If this value is blank, default will be 'defaultUser'.
SNMPV3_USER=

##############################################################################
# Protocol Mediator options
##############################################################################

#
#
# Sub-agent configuration
#
# If the master agent is not being used (i.e. START_MASTER_AGENT=no), then
# setting SUB_AGENT=yes indicates that the mediator should be started with a
# port number argument by snmpdx or a third party master agent. Otherwise, set
# to no if the mediator is to be started with a manually configured port
# number.
#
# If START_MASTER_AGENT=yes then this setting is ignored.
#
SUB_AGENT=yes

#
# Mediator request port. If the START_MASTER_AGENT="no" and SUB_AGENT="no", the
# default is 33000, otherwise it is dynamically allocated.
#
SPAPM_REQ_PORT=

#
# set to yes to enable v3 mediator traps (requires START_MASTER_AGENT=yes and
# START_MEDIATOR=yes)
#
SPAPM_TRAPS_ARE_V3=no

#
# Default port for traps
#
SPAPM_TRAP_PORT=162

# This is also used to define the interface to which SNMP traps will be
# sent by the protocol mediator independently of the setting of
# AGENT_INTERFACE_NAME. If not defined, traps will be issued from the
# host's default interface.
#
SPAPM_TRAP_INTERFACE=
```

**CODE EXAMPLE 11-1** Example of a `spama.conf` File *(Continued)*

```
#
# Agent option flags
#
# -a    Send attribute change notifications
# -s    Send state change notifications
# -c    Send object creation notifications
# -C    Enable object creation notifications during initialization
# -d    Send object deletion notifications
# -l    Enable current problem list logs by default
#
SPAPM_OPTIONS="-ascCdl"
```

# Access Control

Access control is based on the IP address and community of the managers' host machine and the communities specified for each sub-agent. Access rights for communities and host machines are defined in ACL files.

ACL files also define the hosts to which the agent sends traps. When a trap is sent, the agent sends it to all hosts listed in the <*trapInterestHostList*> of the ACL file.

There are three ACL files:

- `spapm.acl` controls access to the Mediator, either directly from management applications or, more normally, from `snmpdx`. It also defines the required recipients of SNMP traps.
- `spapm_snmpdx.acl` controls access to the Mediator when this is configured as a sub-agent of `snmpdx`.
- `spama.acl` controls access through the SNMPv3 Master Agent

Access control, communities and trap forwarding parameters for SNMPv3 are defined in `spama.uacl` and `spama.security`.

# Format of an ACL File

An ACL file contains an `acl` group defining community and manager access rights and a `trap` group defining the community and hosts for sending traps. An ACL file can also contain comment lines, denoted by a hash symbol ( # ) as the first character on the line.

---

**Note –** There are certain differences in the syntax of the `spapm_snmpdx.acl` file. See the comments in the script for details.

---

## The `acl` Group

An `acl` group contains one or more lists of community configurations using the following syntax:

```
acl = {
    <list1>
    <list2>
    ...
    <listN>
}
```

The `acl` group in this file specifies the access rights for specific communities and managers. It comprises a list of community configurations having the following format:

```
{
   communities = <communityList>
   access = <accessRights>
   managers = <hostList>
}
```

The *<communityList>* is a list of SNMP community names to which this access control applies. The community names in this list are separated by commas.

The <*accessRights*> specifies the rights to be granted to all managers running on the machines specified in the managers item. There are two possible values:

- `read-write`
- `read-only`

The <*hostList*> item specifies the host machines of the managers to be granted the access rights. The <*hostList*> is a comma-separated list of hosts, each of which can be expressed as any one of the following:

- A host name (for example, `hubble`)
- An IP address (for example, `123.456.789.12`)
- A subnet mask (for example, `123!255!255!255`)

---

**Note –** To distinguish between IP addresses and subnet masks in an ACL file, each integer in a subnet mask is separated by an exclamation mark (!) instead of a dot.

---

**CODE EXAMPLE 11-2** Example `acl` Group

```
acl = {
    {
        communities = public, private
        access = read-only
        managers = rag, tag, bobtail
    }
    {
        communities = tigger
        access = read-write
        managers = brittas
    }
}
```

# The `trap` Group

The `trap` group specifies the hosts to which the agent can send traps. Configuration is necessary only if the Mediator is required to send SNMPv2 traps, not if the Mediator is to send SNMPv3 traps by means of the SNMP Master Agent.

This group contains one or more trap community definitions using the following syntax:

```
trap = {
    <community1>
    <community2>
   ...
    <communityN>
}
```

Each line defines the association between a set of hosts and the SNMP community string in the traps to be sent to them. Each `trap-community` definition has the following format:

```
{
   trap-community = <trapCommunityString>
   hosts = <trapInterestHostList>
}
```

The <*trapCommunityString*> item specifies the SNMP community string. It is included in the traps sent to the hosts specified in the `hosts` item.

The <*trapInterestHostList*> item specifies a comma-separated list of hosts. Each host must be identified by its name or complete IP address, as shown in the following example:

**CODE EXAMPLE 11-3** Example `trap` Group

```
trap = {
     {
          trap-community = tigger
          hosts = gandalf, frodo
     }
}
```

# Mediator Configuration Files

This section describes the format of the:

“spapm.acl File” on page 105

“spapm_snmpdx.acl File” on page 106

## spapm.acl File

The spapm.acl file defines the access control for the Protocol Mediator. The general format of the file is described in “Format of an ACL File” on page 102. This section contains information relating specifically to the spapm.acl file, and an example of the file is shown in CODE EXAMPLE 11-4.

The file is located, by default, in /etc/opt/SUNWspa.

If an ACL file exists, the access rights it defines apply to all managers or platform agent servers that access the agent through its SNMP adapter. If the ACL file does not exist when the agents are started, all managers are granted full access to the agent through the SNMP adapter and no traps are generated.

To enable access control and traps for the SNMP adapter, ensure that an ACL file exists when any agents are started. As the ACL file contains security-related information, assign it restricted access rights, readable only by root.

The acl and trap groups follow the formats described in “The acl Group” on page 102 and “The trap Group” on page 104, respectively.

If the Mediator is registered as a sub-agent of a master agent (such as snmpdx), you must specify localhost as the manager in the spapm.acl file as this is the origin of SNMP packets forwarded by the master agent. When snmpdx is used, it forwards community strings without change and therefore you must specify the communities listed in this file in the spapm_snmpdx.acl file as well (see CODE EXAMPLE 11-4).

**CODE EXAMPLE 11-4** Example spapm.acl File

```
#
#  @(#)spapm.acl     1.6 03/01/29 SMI
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
#
# Template ACL file for Sun SNMP Management Agent for Sun Fire B1600

acl = {
```

**CODE EXAMPLE 11-4** Example `spapm.acl` File *(Continued)*

```
    {
        communities = public, private
        access = read-only
        managers = rag, tag, bobtail
    }
    {
        communities = tigger
        access = read-write
         managers = brittas
     }
}

trap = {
    {
        trap-community = tigger
        hosts = brittas
    }
}
```

The trap group defines where SNMPv2c notifications are sent.

## `spapm_snmpdx.acl` File

In the default configuration, the Mediator runs as the sub-agent of `snmpdx`. You can modify this file to enable access based on the source hostname. The communities and access rights must match those in the `spama.acl` file.

The `acl` group in this file specifies the access rights for specific communities and managers. It comprises a list of community configurations with the format:

```
#   {
#       communities = <communityList>
#       access = <accessRights>
#       managers = <hostList>
#   }
```

- *communityList* is a comma separated list of community names to which this access control applies.
- *accessRights* specifies the permissions granted to the managers named in the *hostList*.
- *hostList* is a comma-separated list of host names to be granted the specified *accessRights*,

In the first example in CODE EXAMPLE 11-5, the systems `rag`, `tag` and `bobtail` are configured for `read-write` access on the communities `public` and `private`. The system `brittas` is configured for `read-write` access with the community `tigger`.

The second example applies to the configurations using the SNMPv3 Master Agent (`START_MASTER_AGENT=yes` in `spama.conf`) where SNMP packets received by `snmpdx` are seen to have originated on the localhost. Read-only access is configured for the communities `public` and `private`. Read-write access is configured for the community `tigger`. These communities are mapped from SNMPv3 contexts by the Master Agent. Therefore, any SNMPv3 context for which this access is required must have a corresponding community specified in this file.

**CODE EXAMPLE 11-5** Example `spapm_snmpdx.acl` File

```
#  @(#)spapm_snmpdx.acl1.8 03/01/29 SMI
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# Template snmpdx Access Control file for Sun SNMP Management Agent for Sun
# Fire B1600
#
# Example 1:
#
# acl = {
#     {
#         communities = public, private
#         access = read-only
#         managers = rag, tag, bobtail
#     }
#     {
#         communities = tigger
#         access = read-write
#         managers = rag, tag, bobtail
#     }
# }
#
# Example 2:
#
 acl = {
     {
         communities = public, private
         access = read-only
         managers = localhost
     }
     {
         communities = tigger
         access = read-write
         managers = localhost
```

**CODE EXAMPLE 11-5** Example `spapm_snmpdx.acl` File *(Continued)*

```
      }
  }
#
# Trap destinations are defined in spapm.acl and spama.acl.
# This entry does not need to be edited.
trap = {
}
```

# Master Agent Configuration Files

You should configure these files only if you are enabling the Master Agent functionality (`START_MASTER_AGENT=yes` in `spama.conf`).

This section describes the format of the:

- "`spama.acl` File" on page 108
- "`spama.uacl` File" on page 109
- "`spama.security` File" on page 110

## `spama.acl` File

The `spama.acl` file defines the access control for the Master Agent. The general format of the file is described in "Format of an ACL File" on page 102. This section contains information relating specifically to the `spama.acl` file.

The file is located, by default, in `/etc/opt/SUNWspa`.

To enable access control and traps for the SNMP adapter, ensure that an ACL file exists when any agents are started. As the ACL file contains security-related information, assign it restricted access rights, readable only by `root`.

This file defines SNMPv1 and SNMPv2c access permissions, and also the recipients for SNMP notifications. If `SPAPM_TRAPS_ARE_V3=yes` in `spama.conf`, the traps are sent as SNMPv3 traps, otherwise they are sent as SNMPv2c traps.

### acl Group

If you are employing SNMPv3, you will probably want to prohibit write access by SNMPv1 and SNMPv2c. Therefore, this acl typically allows only read-only access.

**CODE EXAMPLE 11-6** Example acl Group

```
acl = {
     {
          communities = public, private
          access = read-only
          manager = localhost
     }
}
```

### trap Group

The trap group for this file follows the format described in “The trap Group” on page 104.

## spama.uacl File

This file is used in conjunction with the spama.security file to enable SNMPv3 security when the Master Agent is active. The general format of the file is described in “Format of an ACL File” on page 102. Additional configuration parameters are explained in this section. An example of the file, with most of the comments removed for clarity, is shown in CODE EXAMPLE 11-7.

The file is located, by default, in /etc/opt/SUNWspa.

### acl Group

The acl group contains the following parameters:

- *context-names*—This is a comma-separated list of context names.
- *access*—The possible values are:
  - read-only
  - read-write

- *security-level*—The possible values are:
  - noAuthNoPrivacy
  - authNoPrivacy
  - authPrivacy
- *users*— This is a comma-separated list of user names.

In the following example, access is granted for defaultUser and any request from defaultUser in the context of public and null with a minimum security of authNoPrivacy is authorized. All other SNMP requests are rejected.

There is no trap group in this file.

**CODE EXAMPLE 11-7** Example spama.uacl File

```
#ident "@(#)spama.uacl 1.4 01/29/03 SMI"
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# This software is the proprietary information of Sun Microsystems, Inc.
# Use is subject to license terms.
#
# Template ACL file
#
 acl = {
     {
         context-names = public,null
         access = read-write
         security-level=authNoPriv
         users = defaultUser
     }
 }
```

## spama.security File

The spama.security file specifies the users that are allowed access to the Master Agent, and the SNMPv3 encryption and authentication keys.

The file comprises a set of userEntry rows having the following format:

```
userEntry=<engine ID>,<user name>,<security name>,<authentication algorithm>,
<authentication key>, <privacy algorithm>, <privacy key>, <storage type>,<template>
```

**Caution –** Do not edit any parameter other than the `userEntry` row in this file.

These fields are explained in TABLE 11-2.

**TABLE 11-2** User Configurable Parameters in `spama.security`

| Parameter | Description |
|---|---|
| engine ID | The ID of the SNMP engine to be used. It can be:<br>■ A hexadecimal string<br>■ a text string representing an `engineID` in the form `<address>:<port>:<IANA number>`<br>■ The string `localEngineID`, which will be suitable in most cases |
| user name | The user name to which this entry applies |
| security name | The security name to be mapped to this user name. Normally, they should be identical. |
| authentication algorithm | The authentication algorithm to be used. This can be one of the following:<br>• `usmHMACMD5AuthProtocol`<br>• `usmHMACSHHAuthProtocol`<br>• `usmNoAuthProtocol` |
| authentication key | The key to be used with the authentication algorithm. This can be one of the following:<br>• A text password (minimum eight characters)<br>• A localized hexadecimal key, for example, `0x0098768905AB67EFAA855A453B665B12` |
| privacy algorithm | The privacy algorithm to be used. This can be one of the following:<br>• `usmDESPrivProtocol`<br>• `usmNoPrivProtocol` (default if unspecified) |
| privacy key | The key to be used with the privacy algorithm. This can be one of the following:<br>• A text password (minimum eight characters)<br>• a localized hexadecimal key, for example, `0x0098768905AB67EFAA855A453B665B12` |
| storage type | The only acceptable value is `3`, which is the default if unspecified. |
| template | The default is `false` (it is not necessary to change this value) |

An example of the file, with most of the comments removed for clarity, is shown in CODE EXAMPLE 11-8.

The file is located, by default, in `/etc/opt/SUNWspa`.

The default spama.security file contains two sample users that you can modify and uncomment to define your own users. The first specifies a user named `defaultUser` with:

- Authentication using the MD5 algorithm only
- No privacy
- The authentication password "`mypassword`"

The second specifies a user named `defaultUser` with:

- Authentication using the MD5 algorithm and the authentication password "`mypassword`"
- Privacy using the DES algorithm and the privacy password "mypassword"
- Privacy using the DES algorithm

**CODE EXAMPLE 11-8** Example `spama-security` File

```
#ident  "@(#)spama.security 1.7 01/29/03 SMI"
#
# Copyright 2002 Sun Microsystems, Inc. All rights reserved.
# This software is the proprietary information of Sun Microsystems, Inc.
# Use is subject to license terms.
#
# Template security file

# localEngineBoots=0

# defaultUser configuration. Authentication only.
# userEntry=localEngineID,defaultUser,,usmHMACMD5AuthProtocol,mypasswd

# defaultUser configuration. Authentication and encryption.
# userEntry=
localEngineID,defaultUser,null,usmHMACMD5AuthProtocol,mypasswd,usmDESPrivProt
ocol,mypasswd,3,
```

CHAPTER 12

# Configuring the Software

This chapter describes the default configuration after installation, and explains how to modify the files described in Chapter 11.

The chapter contains the following sections:



## Default Configuration

The software installs with the following default configuration:

- The Master Agent is disabled (`START_MASTER_AGENT=no`).
- The Mediator is enabled (`START_MEDIATOR_AGENT=yes`).
- The Mediator is configured as a sub-agent of `snmpdx`.

**Note –** For security reasons, you should configure the `snmpdx` ACL file to restrict access to exclude all systems other than those monitoring the agent.

### Access Control

To enable access control for the Mediator, configure the Mediator ACL file as described in "`spapm.acl` File" on page 105.

If you are using `snmpdx` (default configuration), modify `spapm_snmpdx.acl` to set up access permissions and `spapm.acl` to set up trap recipients.

## Starting and Stopping the Mediator

Start the Mediator using the normal snmpdx startup up script:

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

Stop the Mediator using the Mediator script:

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

# Manual Configuration for Direct Access

As `snmpdx` supports only SNMPv1, if you wish to use SNMPv2c specific `get-bulk` operations and do not wish to use the Master Agent, you can configure the port used by the Mediator to enable direct SNMPv2c access.

To configure the Mediator manually, make the following changes in `spama.conf`:

1. **Set** `SUB_AGENT=no`**.**
2. **Set** `SPAPM_REQ_PORT` **to the required port number.**

   SNMPv2c requests sent to the Mediator must be sent to this port.

## Mediator as a Sub-Agent of a Third-Party Master Agent

To configure the Mediator as a sub-agent of a third-party master agent that supports specification of a port number by means of a command line parameter:

1. **Configure the Mediator ACL file to allow access from the localhost (see “`spapm.acl` File” on page 105).**
2. **Configure your master agent to start the Mediator with the following invocation, using an appropriate port number:**

```
/etc/init.d/spama start <port>
```

3. **Configure the master agent to forward requests to the following OID sub trees**:
   - .iso.org.dod.internet.mgmt.mib-2.entityMIB
   - .iso.org.dod.internet.private.enterprises.sun.products.sunFire.sunPlatMIB

   or numerically:
   - .1.3.6.1.2.1.47
   - .1.3.6.1.4.1.42.2.70.101

# Mediator and the SNMPv3 Master Agent

To enable the Mediator and Master Agent you must make the following changes as a minimum:

1. **In the spama.conf file**:
   a. **Set** `START_MASTER_AGENT=yes`.
   b. **Set** `SUB_AGENT=no`.
2. **Configure the Mediator ACL file to enable access from the localhost as described in “`spapm.acl` File” on page 105.**
3. **Configure snmpdx to enable access from localhost as described in “`spapm_snmpdx.acl` File” on page 106.**
4. **Configure the Master Agent ACL file to enable access from the desired managers as described in “`spama.acl` File” on page 108.**
5. **Configure the security files to define SNMPv3 users, context, and authentication and encryption levels as described in “`spama.acl` File” on page 108 and “`spama.security` File” on page 110.**

## Starting and Stopping the Agents

Start the Mediator and Master Agent using the Mediator script:

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

Stop the Mediator and Master Agent using the Mediator script:

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

## Forwarding SNMPv3 Traps

To configure the Master Agent to forward SNMPv3 traps from the Mediator, in the spama.conf file (see “spama.conf” on page 92):

1. **Set** SPAPM_TRAPS_ARE_V3=yes**.**
2. **Optionally, set the** SNMPV3_USER**.**

---

**Note –** The trap user must be configured as an SNMPV3 user in spama.uacl and spama.security.

---

CHAPTER 13

# Uninstalling the Software

This chapter explains how to uninstall 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, and re-enables `snmpdx`.

Configuration changes made automatically by the SNMP software are restored to their original state. However, if you modified any external file settings, such as the snmpdx ACL file, you must restore these manually after removing the SNMP software.

**Note –** The procedures listed below do not uninstall the Java SNMP API package, `SUNWjsnmp`. If you want to reinstall the Solaris version of this package, you must first remove the Java SNMP API.

## Platform Agent and Target Agent Packages

To remove the platform agent packages from the platform agent server, type:

```
# pkgrm SUNWbgpmr SUNWbgpm SUNWjdrt SUNWjsnmp SUNWbgpjo \
SUNWbgodr SUNWbgod SUNWbgcmr SUNWbgcm SUNWbgpc SUNWbgptk
```

**Caution –** Remove the `SUNWjdrt` and `SUNWjsnmp` packages with caution as they are both system packages and may be used by other products.

To remove the Target Platform packages from the Sun Fire B100s blade(s), type:

```
# pkgrm SUNWbgpr SUNWbgcm SUNWbgpc SUNWbgptk
```

# Domain Agent Packages

To remove the Domain Agent packages from the Sun Fire B100s blade(s), type:

```
# pkgrm SUNWbgpmr SUNWbgpm SUNWjdrt SUNWjsnmp SUNWbgpji \
SUNWbgidr SUNWbgcmr SUNWngcm SUNWbgpc SUNWbgptk
```

**Caution –** Remove the `SUNWjdrt` and `SUNWjsnmp` packages with caution as they are both system packages and may be used by other products.

CHAPTER 14

# Troubleshooting

This chapter provides information to help you troubleshoot your system.

*Problem*

- **There is no response from the SNMP agent when using the default configuration (**`snmpdx`**).**

1. **Make sure the Mediator is running by typing:**

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
```

If the response is similar to that above, the Mediator process is running.

Stop the Mediator and restart it by typing:.

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

2. **Make sure the correct version of Java is installed by typing:**

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

This should report version 1.4 or later. If this is not the case. install the Java 1.4 JDK as described in “Java Environment” on page 70.

3. **Make sure the correct version of** `SUNWjsnmp` **is installed by typing:**

```
# pkginfo -l SUNWjsnmp | grep VERSION
   VERSION:  5.0
```

If version 1.0 is shown, remove the `SUNWjsnmp` package and re-install the version in the `SUNWspa.*.tar.Z` archive (see "Java SNMP API" on page 71).

4. **Make sure the spama.conf file contains the following entries:**

`START_MASTER_AGENT=no`

`START_MEDIATOR=yes`

`SUB_AGENT=yes`

5. **Make sure the Mediator is registered correctly with** `snmpdx` **by typing:**

```
# cat /var/snmp/snmpdx.st
spapm spapm 2516 34050
snmpd snmpd 2567 34053
```

The `spapm` entry above shows the Mediator is registered as a sub-agent of `snmpdx`.

6. **Make sure that** `/etc/snmp/conf/spapm.reg` **and** `/etc/snmp/conf/spapm.rsrc` **are not corrupted.**

Stop the Mediator and restart it by typing:

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

7. **Make sure that the permissions in the ACL files are set correctly.**
   - `spapm_snmpdx.acl` defines access for SNMP manager being used.
   - `spapm.acl` defines access for `localhost`.

   For more information, see Chapter 11.

### *Problem*

- **There is no instrumentation for Hard Disk Drive (HDD) or Ethernet MAC addresses when using the platform agent.**

  This information is available only when the operating environment is running on the target Sun Fire B100s blade.

1. **Make sure that the Sun Fire B100s blade is booted.**

2. **Make sure the target instrumentation is running on the Sun Fire B100s blade by typing:**

```
# netstat -an | grep 1099
      *.1099        *.*         0      0 24576      0 LISTEN
```

If no port is listening, the target instrumentation is not running.

Start the instrumentation on the Sun Fire B100s blade by typing:

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

3. **Make sure the correct version of Java is installed by typing:**

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

This should report version 1.4 or greater. An incorrect version can cause the instrumentation to start, but then fail after a short period.

If this is not the case. install the Java 1.4 JDK as described in “Java Environment” on page 70.

### *Problem*

- **The agent is accessible, but there is no instrumentation for the monitored platforms**

1. **Make sure the discovery daemon is running by typing:**

```
# netstat -a | grep mismi
      *.mismi              *.*                0      0 24576      0 LISTEN
      *.mismi                           *.*  0
0 24576      0 LISTEN
```

The above output indicates that the discovery daemon is listening for requests from the managed platform(s).

**a. Make sure that** `/etc/services` **has the following entry:**

```
mismi           8265/tcp                # MISMI Discovery
```

**b. Make sure that** `/etc/inetd.conf` **has the following entry:**

```
# MISMIDISCOVERY - mismiDiscovery daemon
mismi   stream  tcp6    nowait  root    /opt/SUNWspa/bin/mismiDiscovery
mismiDiscovery
```

**c. Make sure that** `/etc/inetd.conf` **is a symbolic link to** `/etc/inet/inetd.conf` **by typing:**

```
# ls -l /etc/inetd.conf
lrwxrwxrwx   1 root     root          17 Jan  7 17:08 /etc/inetd.conf ->
./inet/inetd.conf
```

If the link does not exist, the update of the file will fail during the installation of the `SUNWbgodr` package.

Correct the `inetd` configuration and restart by typing:

```
# pkill -1 inetd
```

**2. Make sure the platform has been discovered by typing:**

```
# netstat -a | grep mismi
      *.mismi              *.*                0      0 24576      0 LISTEN
blade-174-119.36780  hornet-sc.mismi       8192      0 24820      0 ESTABLISHED
      *.mismi                           *.*  0
0 24576      0 LISTEN
```

The output shows that the discovery daemon is listening and that a connection to the platform system controller (called <`hornet-sc`>) has been established.

If no connection is present, check the system controller setup as described in “Configuring the System Controller” on page 85.

*Problem*

- **SNMPv3** `get` **and** `set` **requests time out.**
  - **Possible cause**

  Either the `localEngineId` or the number of `localEngineBoots` in the `spama.security` file may have been edited or deleted.

  - **Check**

  It is not straightforward to determine if the file has been edited.

  - **Fix**

  Restart the agent as shown below, and then restart the management application to resynchronize them.

```
# /etc/init.d/spama stop
# /etc/init.d/spama start
```

*Problem*

- **SNMP** `get` **and** `set` **requests time out**
  - **Possible cause**

  On a heavily loaded system it is possible for the `snmpdx` master agent to timeout its requests to the SNMP Mediator. This timeout is currently set to 2s (2000000 s).

  - **Check**

  It is not straightforward to determine if the timeout reported by a management application has occurred between `snmpdx` and the SNMP Mediator, or between the management application and `snmpdx`.

  - **Fix**

  You can increase the timeout by editing the timeout property in the `/etc/snmp/conf/spapm.reg` file. If you edit the file, restart the Mediator by typing:

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

APPENDIX **A**

# Installing J2RE 1.4 to Co-exist with J2SE 1.3.1

This appendix describes how to install Java 2 Runtime Environment (J2RE) Standard Edition 1.4 to co-exist with J2SE 1.3.1 on the platform agent server and B100s domains, and how to modify the startup scripts to locate the installation.

The appendix contains the following sections:



## Installing J2RE 1.4

To install J2RE 1.4 to co-exist with J2SE 1.3.1, as discussed in "Java Environment" on page 70, follow the procedure below.

J2RE 1.4 is available as a self-extracting binary file from:

`http://java.sun.com/j2se/1.4/download.html`

Follow the instructions below to install J2RE. Further information on downloading the file is available on the above web site.

---

**Note –** This product requires only 32-bit support, so it is not necessary to install the 64-bit supplement for the J2RE.

---

In the following steps, substitute the appropriate J2RE update version number in the following steps for <ver>.

For example, if you are downloading update 1.4.0_01, the following command:

```
# chmod +x j2re-1_4_<version>-solaris-sparc.sh
```

becomes:

```
chmod +x j2re-1_4_0_01-solaris-sparc.sh
```

1. **Download and check the file size**

   The required file is:

   j2re-1_4_<ver>-solaris-sparc.sh

   Before you download a file, note its size, which is provided on the download page. Once the download has completed, check that you have downloaded the full, uncorrupted software file.

   Make sure that you download the file to a location accessible to root (for example, in /tmp).

2. **Become root by running** su **and entering the super-user password.**

3. **Make sure that execute permissions are set on the self-extracting binary.**

```
# chmod +x j2re-1_4_<ver>-solaris-sparc.sh
```

4. **Change to the directory where the files must be installed.**

```
# cd /usr
```

5. **Run the self-extracting binary.**

   A directory is created called /usr/j2re1.4.<ver>, which contains the J2RE.

6. **Make sure that the J2RE has installed correctly.**

```
# /usr/j2re1.4.1_01/bin/java -version
java version "1.4.1_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_01-
b01)
Java HotSpot(TM) Client VM (build 1.4.1_01-b01, mixed mode)
```

This should report version `1.4` or later. This example shows that the version is `1.4.1_01`.

7. **Delete the self-extracting binary.**

8. **Exit the root shell.**

# Editing the Startup Scripts

This section explains how to modify the startup scripts when you install J2RE 1.4. The modifications

Read this section in conjunction with Chapter 10.

## Domain Hardware Monitoring

These steps relate to Step 4 in “Installing Software for Domain Hardware Monitoring” on page 79.

1. **On each monitored B100s blade, edit each of the following startup scripts:**
   - `/etc/init.d/spaibdm`
   - `/etc/init.d/spapom`

   by replacing the line that reads

   `JAVA=/usr/j2se/bin/java`

   with

   `JAVA=/usr/j2re1.4.<ver>/bin/java`

2. **On each monitored B100s blade, edit the following startup script:**

   - `/etc/init.d/spama`

   by replacing the line that reads

   `JAVA_JAVA=/usr/j2se/bin/java`

   with

   `JAVA_JAVA=/usr/j2re1.4.<ver>/bin.java`

## Platform Hardware Monitoring

Step 1 and Step 2 relate to Step 4 in “To Install the Software with Target Instrumentation” on page 81, and Step 4 in “To Install the Software Without Target Instrumentation” on page 83.

Step 3 relates only to Step 11 “To Install the Software Without Target Instrumentation” on page 83.

1. **On the platform agent server, edit each of the following startup script:**

   - `/etc/init.d/spapom`

   by replacing the line that reads

   `JAVA=/usr/j2se/bin/java`

   with

   `JAVA=/usr/j2re1.4.<ver>/bin/java`

2. **On the platform agent server, edit the following startup script:**

   `/etc/init.d/spama`

   by replacing the line that reads

   `JAVA_JAVA=/usr/j2se/bin/java`

   with

   `JAVA_JAVA=/usr/j2re1.4.<ver>/bin.java`

3. **On each monitored B100s blade (target), edit the following startup script:**
   - `/etc/init.d/spardp`

   by replacing the line that reads

   `JAVA=/usr/j2se/bin/java`

   with

   `JAVA=/usr/j2re1.4.<ver>/bin/java`

# Index

## E



## F



## G



## H



## I

## Q



## R



## S

## T



## U



## W