Netra™ CT Server Software Developer's Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 816-2486-11
August 2004, Revision A

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

---

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

The *Netra CT Server Software Developer's Guide* contains information for developers writing application software for the Netra™ CT 810 and 410 servers. This manual assumes you are a software developer familiar with UNIX commands and networking applications.

How This Book Is Organized

Chapter 1 contains an overview of the Netra CT software and lists the requirements for developing software applications for the platform.

Chapter 2 displays the system's various equipment models. The diagrams in this chapter demonstrate how the Netra CT software views the hardware components.

Chapter 3 offers a tutorial in writing applications that interface with the Netra CT server software.

Chapter 4 introduces the application programming interfaces for the Netra CT server including the Netra CT element management agent software.

Chapter 5 describes the Netra CT Simple Network Management Protocol (SNMP) management information base (MIB).

Chapter 6 presents the design of the Netra CT software modules and how they relate to each other.

Chapter 7 provides an overview of the Netra CT Processor Management Services (PMS) software.

Chapter 8 defines the Solaris™ operating system's platform information and control library (PICL) software and how you can use it to set the watchdog timer.

For obscure or difficult terminology definition, see the Glossary.

Typographic Conventions

Typeface[*]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`

`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

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

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

The Netra CT Server documentation is listed in the following table.

Title	Part Number
Netra CT Server Start Here	816-2479
Netra CT Server Product Overview	816-2480
Netra CT Server Installation Guide	816-2481
Netra CT Server Service Manual	816-2482
Netra CT Server System Administration Guide	816-2483
Netra CT Server Safety and Compliance Manual	816-2484
Netra CT Server Software Developer's Guide	816-2486
Netra CT Server Release Notes	817-0939

You might want to refer to documentation on the following software for additional information: the Solaris operating environment, the ChorusOS™ environment, OpenBoot™ PROM firmware, and the Netra High Availability (HA) Suite.

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:

Netra CT Server Software Developer's Guide, part number 816-2486-11

CHAPTER 1

Programming Environment

This chapter provides an overview of the software environment that forms the basis for developing applications for the Netra CT server:

- "Netra CT Server" on page 1
- "Hardware Description" on page 1
- "Software Description" on page 3

Netra CT Server

The Netra CT server system consists of a host CPU, an alarm card which is the nexus of system management, optionally one or several satellite CPUs, and one or several CompactPCI (cPCI) I/O cards. Different software combinations run on each of these elements as is shown in FIGURE 1-1.

Hardware Description

This section provides brief descriptions of the Netra CT server board components and hot-swapping capabilities. See the *Netra CT Server Product Overview* (816-2480) for more information.

Alarm Card

An alarm card is used in the Netra CT 810 and Netra CT 410 servers to control system functions. The board is plugged into slot 8 for the Netra CT 810, and into slot 1 for the Netra CT 410 server. ChorusOS 5.0 is the operating system running on the alarm card, and the boot environment is controlled by boot control firmware. Developers use a command-line interface (CLI) to provide an administrative interface to the system. Drawer-level monitoring and control of the system is accomplished through Managed Object Hierarchy (MOH) and Processor Management Service (PMS) software.

Host CPU Board

The host CPU board is the same for both Netra 810 and Netra CT 410 servers. The board is plugged into slot 1 for the Netra CT 810, and into slot 3 for the Netra CT 410 server. The Solaris OS runs on these boards. MOH and PMS provide local and drawer-level monitor and control functions.

Satellite CPU Boards

Several satellite CPU cards can occupy the I/O slots and perform normal CPU functions independently. MOH and PMS provide local monitor and control functions.

I/O Boards

One or more cPCI boards can occupy I/O slots. The I/O boards are controlled by the host CPU and the Solaris OS running on the host CPU board.

Hot-Swapping Capabilities

Boards and other field-replaceable units (FRUs) can be swapped while the system is running, depending on whether or not they conform to the Hot Swap Specification PICMG 2.1 R 2.0. This ability to hot swap is a feature that is controllable by software if the board itself is hot-swap compliant. For further information on hot swap issues, see the *Netra CT Server Product Overview* (816-2480), *Netra CT Server System Administration Guide* (816-2483), and *Netra CT Server Service Manual* (816-2482).

Software Description

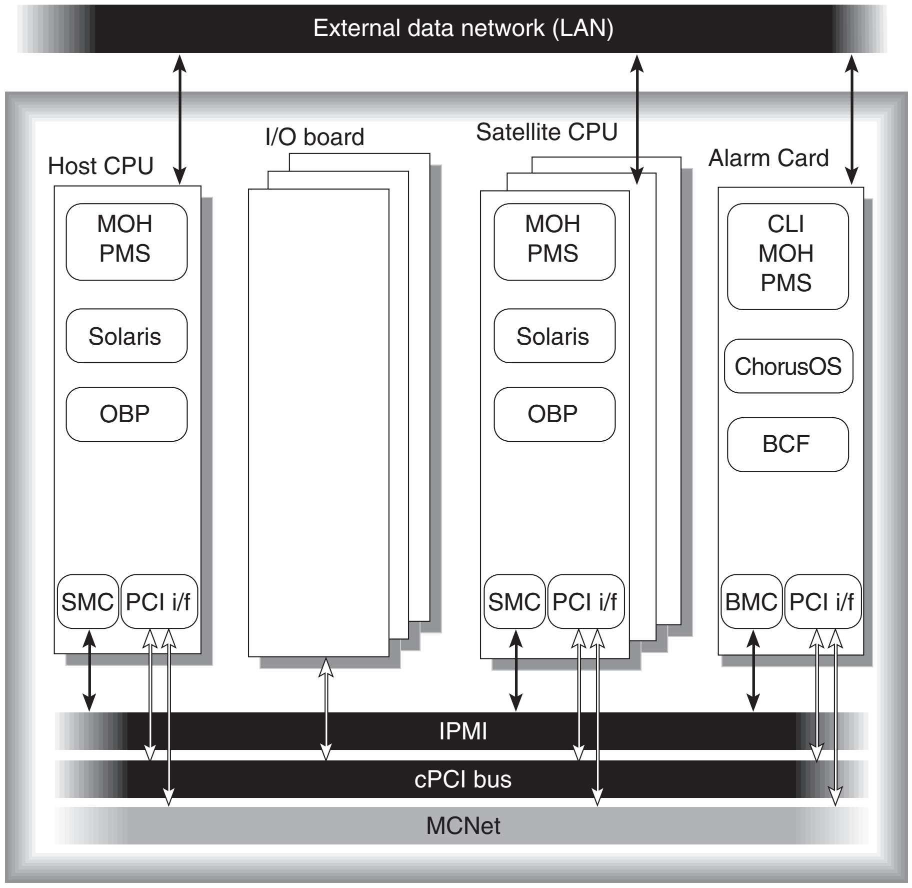


FIGURE 1-1 Netra CT Server Software

The abbreviations shown in FIGURE 1-1 are identified in TABLE 1-1.

TABLE 1-1 Netra CT Server Software Overview

Abbreviation	Name	Description
Solaris	Solaris Operating Environment	Installed by the user. Runs on the host CPU card and on any satellite CPU cards.
ChorusOS	ChorusOS operating environment	Factory-installed on the alarm card. Manages all elements of the Netra CT server that are connected to the midplane.
CLI	Command-line interface	The primary user interface to the alarm card.
MOH	Managed Object Hierarchy	Application that manages the hardware and software components of the system.
PMS	Processor Management Service	Manages processor elements used by client applications.
OBP	OpenBoot PROM firmware and diagnostics	Boot firmware and diagnostics on CPU cards.
BCF	Boot control firmware	Firmware on the alarm card to control booting.
BMC	BMC firmware	Baseboard management controller of the IPMI Controller on the alarm card, which provides a command nexus between Satellite CPU and RMC client during hot swap unconfiguration operations.
SMC	SMC firmware	System Management Controller firmware is related to IPMI Controller on CPU cards. SMC APIs provide client access to local resources such as temperature sensors, watchdog subsystems, and local I2C bus devices; and access to IPMI bus devices.
IPMI	IPMI	Intelligent platform management Interface is a communication channel over the cPCI backplane.
MCNet	MCNet	MCNet is a PICMG 2.14 communication protocol over the cPCI backplane. It can be used to communicate between the alarm card, the host CPU card, and any satellite CPU cards which are MCNet capable.

Operating System Specifics

ChorusOS on the alarm card provides chassis management features that support real-time, multi-threaded applications, and POSIX interfaces to support easy porting of POSIX/UNIX (Solaris) applications. For details of ChorusOS 5.0, refer to the ChorusOS documentation.

Solaris 9 OS on the host and satellite CPU cards provides APIs such as platform information and control library (PICL), reconfiguration coordination manager (RCM), and `cfgadm` (1M), as explained in Chapter 8. The kernel layer interacts with device drivers to control hardware components of the system such as the CPU cards and the I/O boards. These device drivers bind to the kernel using the device driver interfaces (DDI) and driver kernel interfaces (DKI).

Managed Object Hierarchy

The Managed Object Hierarchy (MOH) is a distributed management application that runs on the alarm card, and host and satellite CPUs. MOH on the alarm card provides drawer-level monitoring of the system. MOH on the CPUs, both host and satellite, provides local views of the board on which it runs, and collaborates to provide the status of its components to the MOH on the alarm card. The various MOHs communicate with one another over MCNet. MOH is discussed further in Chapter 6.

Processor Management Services

Processor management services (PMS) software is an extension to the Netra CT platform services software that addresses the requirements of high-availability application frameworks. PMS software enables client applications to manage the operation of the processor CPU board elements within a single Netra CT system or within a cluster of multiple Netra CT systems.

PMS ensures high availability by monitoring a processor element's fault condition, such as OS hangs, deadlock, and panic. The alarm card provides a server-level view showing the state of each CPU card as a plug-in unit. PMS services are enabled separately on the alarm card and on the host CPU. PMS services are discussed further in Chapter 7.

Multicomputing Network

MCNet uses the cPCI backplane on the Netra CT platform to provide Ethernet-like interface to the CPU cards and the alarm card.

Solaris MCNet driver provides standard DLPI v2 interface to higher level protocols and applications. It appears like any other network interface in Solaris when plumbed.

Platform Information Control Library

This Solaris library provides a method for publishing platform-specific information that clients can access in a way that is not specific to the platform. PICL is discussed further in Chapter 8.

Management Framework

The Java Dynamic Management™ Kit (JDMK) development package provides a framework of managed objects and their associated interfaces. SNMP uses a management information base (MIB), which defines managed objects for the elements within the Netra CT server platform. The managed objects are abstract representations of the resources and services within the system. The following interfaces can be used to manage Netra CT system.

SNMP/MIB Support

The `netract` agent supports the following parts of the MIB:

- System group from MIB II
- Interface group from interface MIB
- Physical entity group from ENTITY-MIB

SNMP Interface

The `netract` agent operates on the alarm card, the system host CPU card, and the satellite CPUs in a distributed manner. They all provide the SNMP interface version 2, and Netra CT-specific instrumentation monitoring.

RMI Interface

The `netract` agent uses JDMK service to support common client/server protocols. These include Remote Method Invocation (RMI) which is the mechanism used to support remote, or distributed access to the managed object hierarchy (MOH).

Developing Applications Using PMS

PMS can run on both the alarm card, and host and satellite CPUs. To develop applications that use PMS on the alarm card you need Solaris 9 OS, ChorusOS 5.0, C compiler version, PMS API, and libraries as described in Chapter 7.

To develop applications that use PMS on host and satellite CPUs you need Solaris 9 OS, C compiler version, PMS API, and libraries as described in Chapter 7.

For more information about ChorusOS refer to the *ChorusOS 5.0 Features and Architecture Overview* (806-6897).

Developing Applications to Interface with MOH or SNMP

To develop applications to interface with MOH or SNMP, you need the Solaris 9 OS, Java Virtual Machine (JVM) and the Java Dynamic Management Kit (JDMK) and the Netra CT agent library. For more information about JDMK refer to *Java Dynamic Management Kit 4.2 Tutorial* (806-6633).

Developing Applications to Run on Host or Satellite CPU Boards

To develop applications to run on host or satellite CPU cards you require Solaris 9 OS to access services such as dynamic reconfiguration (DR) framework, and platform information and control library (PICL) API. Standard Solaris tools such as the `cfgadm`(1) command enable service operations such as configuring and unconfiguring system FRUs.

CHAPTER **2**

Netra CT System Equipment Models

This chapter provides illustrations of the Netra CT system equipment models, and contains the following sections:

Modeling a Netra CT System

Equipment models show how the Netra CT element management agent software views the Netra CT system hardware. Each equipment model presents a Netra CT system in a containment hierarchy of hardware components, with the midplane at the root of the hierarchy. For example, a cPCI slot might contain an alarm card, which in turn will contain a number of Ethernet and serial ports. These relationships extending from the midplane form a hierarchy of hardware resources. This hierarchy is modeled using relationships between managed objects representing the hardware resources.

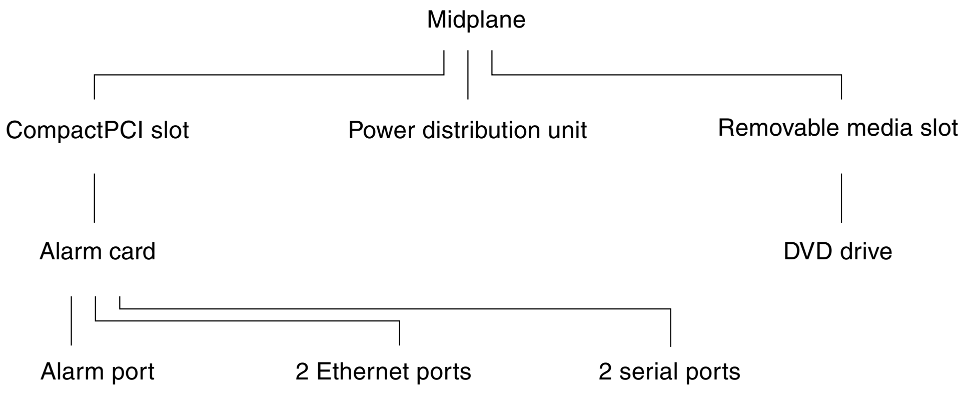


FIGURE 2-1 Partial Hardware Resource Hierarchy

Managed Objects

In the Netra CT software, a managed resource is represented as a managed object, which presents information needed to manage the resource. A managed resource can be represented by a single managed object, or by several managed objects. An agent typically contains or provides views of many managed objects.

FIGURE 2-2 shows the class names of the hardware Netra CT software managed objects, and TABLE 2-1 provides definitions for these objects.

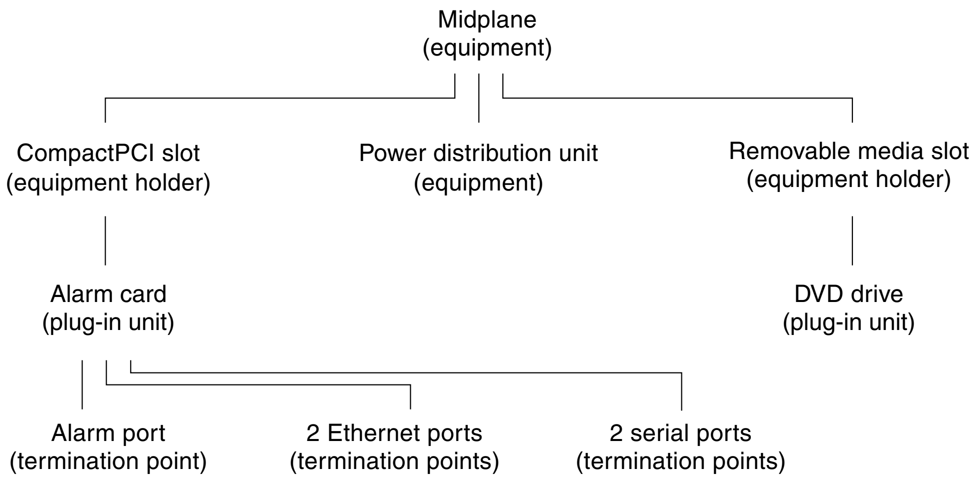


FIGURE 2-2 Hardware Resource Hierarchy Showing Managed Object Classes

TABLE 2-1 Managed Object Class Definitions

Managed Object Class	Definition
Network element	Network elements can be standalone devices or multi-component, geographically distributed systems.
Equipment holder	Represents physical resources of the network element that are capable of holding other physical resources. For example, CompactPCI slots, fan tray slots, and system controller board slots are equipment holder resources.
Plug-in unit	This managed entity represents equipment that can be physically inserted or removed from slots of the system (for example, CompactPCI I/O cards and power supply units).
Equipment	Equipment represents those externally manageable physical components which are not FRUs (for example, a power distribution unit or a CPU temperature sensor) of a network that are not modeled as a plug-in unit or an equipment holder.
Termination point	Represents the points where physical paths terminate (for example, Ethernet and serial ports) and physical path functions.

Viewing the Equipment Model Hierarchies

Both the SNMP interface and the JMX compatible Netra CT element management API provide ways to traverse the equipment containment hierarchy. You can view the managed objects of a Netra CT system through the system's alarm card or through the host CPU board. You can also view the managed objects from the agent on any satellite CPU board. In both system-wide views, the system's midplane is at the top of the equipment hierarchy and all other hardware objects (slots, fan trays, I/O cards, and so on) are displayed subordinate to the midplane.

When viewing the system through the alarm card (defined as the *system view from the alarm card*), the alarm card's termination points (alarm port, Ethernet ports, and serial ports) are displayed in the model, but the host CPU board's termination points are not displayed.

Conversely, when you view the system through the host CPU board (the *system view from host CPU board*), the alarm card's termination points are not displayed, but the host CPU board's termination points, and any hardware connected to the host CPU board (for example, SCSI devices), is displayed.

You can also view the equipment model with the host CPU board or a supported satellite CPU board as the network element at the top of the hierarchy. In these models (defined as the *host CPU board local view* and *satellite CPU board local view*), only the objects directly controlled by the host or satellite CPU board are displayed. Other objects, like the midplane, alarm card, and the power distribution unit, are not seen in these equipment models.

"Netra CT 810 System Equipment Models" on page 12 and "Netra CT 410 System Equipment Models" on page 20 present the equipment models for the Netra CT front and rear-access systems. These sections contain the equipment models shown in the system alarm card view, the host CPU board view, and the host and satellite CPU board views.

Netra CT 810 System Equipment Models

This section contains the following equipment models of the Netra CT 810 server:

- "Rear-Access Netra CT 810 System View From Alarm Card" on page 13
- "Front-Access Netra CT 810 System View From Alarm Card" on page 14
- "Rear-Access Netra CT 810 System View From Host CPU Board" on page 15
- "Front-Access Netra CT 810 System View From Host CPU Board" on page 16
- "Front-Access Netra CT 810 System Host CPU Board Local View" on page 17
- "Rear-Access Netra CT 810 System Satellite CPU Board Local View" on page 19
- "Netra CT 810 System Satellite CPU Board Local View" on page 19

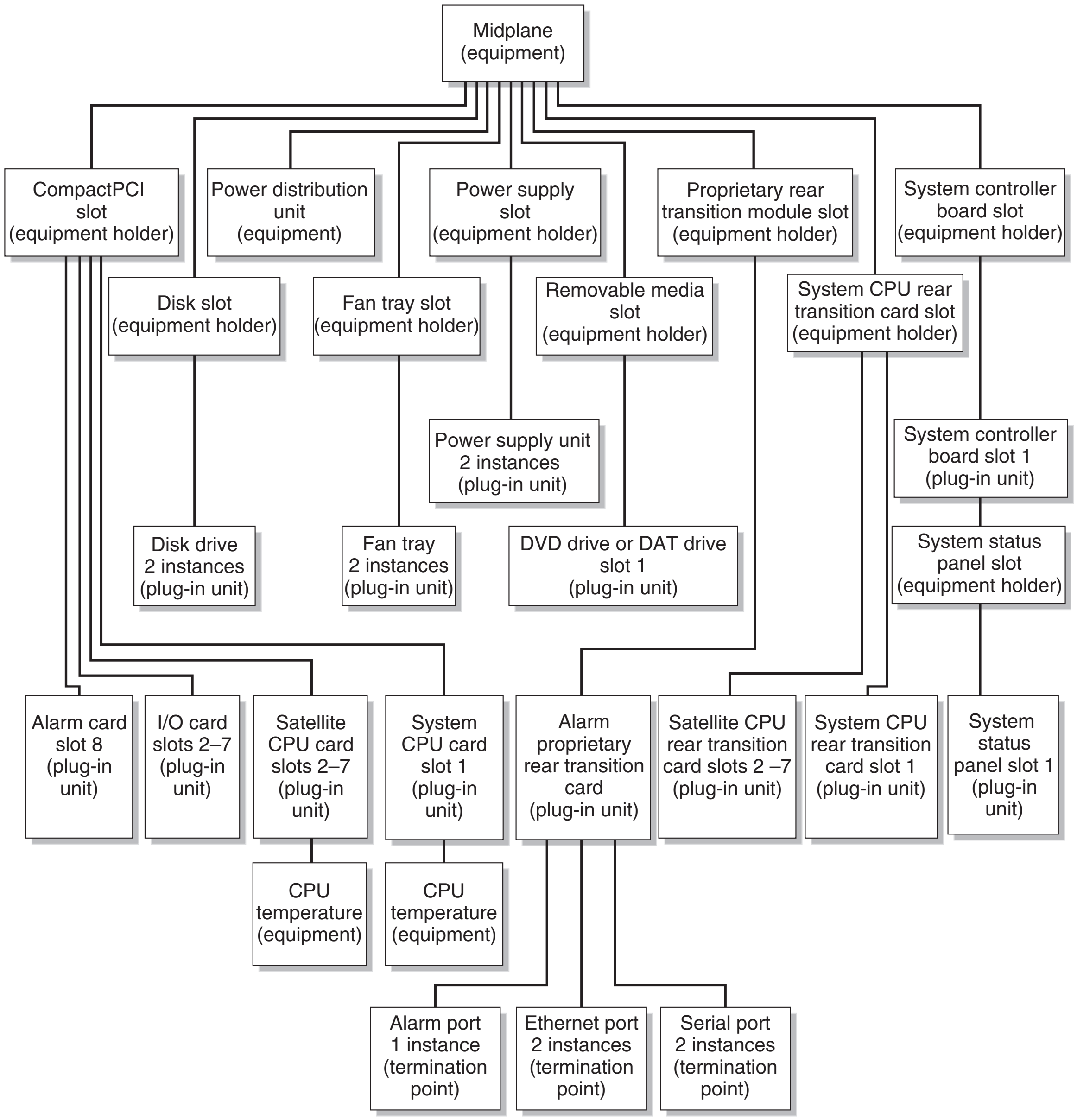

Midplane (equipment)
CompactPCI slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Proprietary rear transition module slot (equipment holder)
System controller board slot (equipment holder)
Disk slot (equipment holder)
Fan tray slot (equipment holder)
Removable media slot (equipment holder)
System CPU rear transition card slot (equipment holder)
Power supply unit 2 instances (plug-in unit)
System controller board slot 1 (plug-in unit)
Disk drive 2 instances (plug-in unit)
Fan tray 2 instances (plug-in unit)
DVD drive or DAT drive slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card slot 8 (plug-in unit)
I/O card slots 2–7 (plug-in unit)
Satellite CPU card slots 2–7 (plug-in unit)
System CPU card slot 1 (plug-in unit)
Alarm proprietary rear transition card (plug-in unit)
Satellite CPU rear transition card slots 2 –7 (plug-in unit)
System CPU rear transition card slot 1 (plug-in unit)
System status panel slot 1 (plug-in unit)
CPU temperature (equipment)
CPU temperature (equipment)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)

FIGURE 2-3 Rear-Access Netra CT 810 System View From Alarm Card

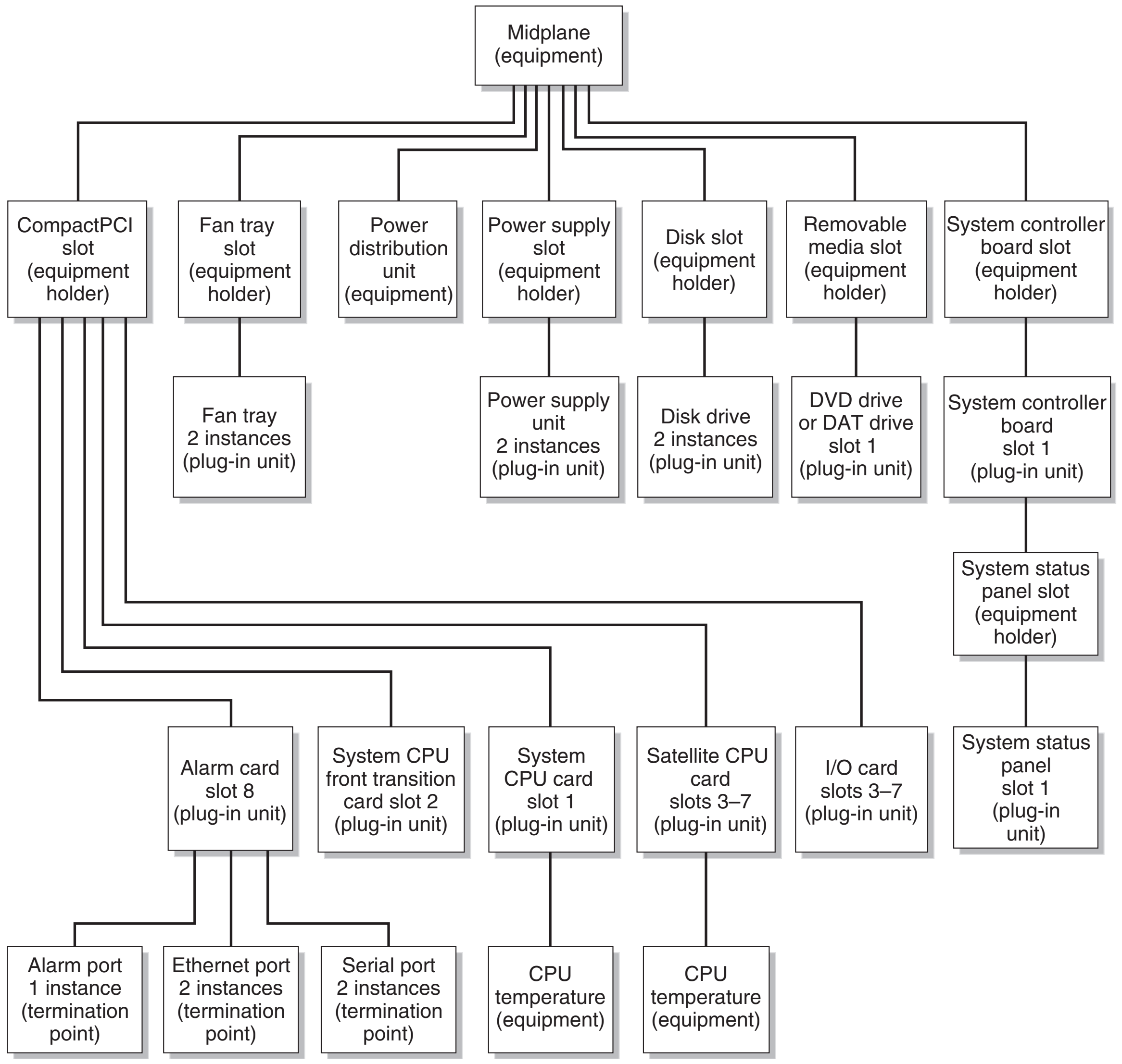
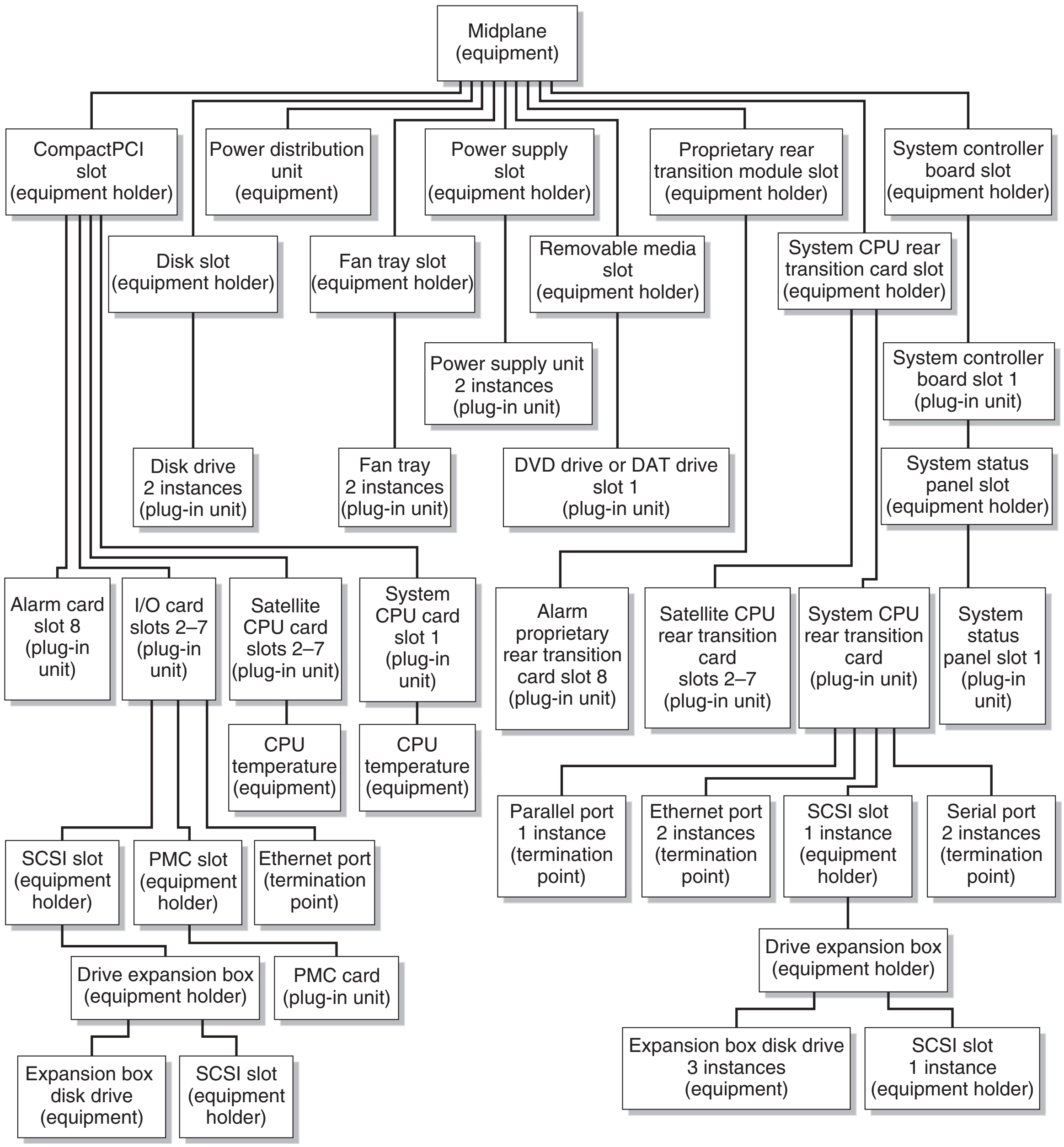

Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Disk slot (equipment holder)
Removable media slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 2 instances (plug-in unit)
Disk drive 2 instances (plug-in unit)
DVD drive or DAT drive slot 1 (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card slot 8 (plug-in unit)
System CPU front transition card slot 2 (plug-in unit)
System CPU card slot 1 (plug-in unit)
Satellite CPU card slots 3–7 (plug-in unit)
I/O card slots 3–7 (plug-in unit)
System status panel slot 1 (plug-in unit)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)
CPU temperature (equipment)
CPU temperature (equipment)

FIGURE 2-4 Front-Access Netra CT 810 System View From Alarm Card



Midplane (equipment)
CompactPCI slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Proprietary rear transition module slot (equipment holder)
System controller board slot (equipment holder)
Disk slot (equipment holder)
Fan tray slot (equipment holder)
Removable media slot (equipment holder)
System CPU rear transition card slot (equipment holder)
Power supply unit 2 instances (plug-in unit)
System controller board slot 1 (plug-in unit)
Disk drive 2 instances (plug-in unit)
Fan tray 2 instances (plug-in unit)
DVD drive or DAT drive slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card slot 8 (plug-in unit)
I/O card slots 2–7 (plug-in unit)
Satellite CPU card slots 2–7 (plug-in unit)
System CPU card slot 1 (plug-in unit)
Alarm proprietary rear transition card slot 8 (plug-in unit)
Satellite CPU rear transition card slots 2–7 (plug-in unit)
System CPU rear transition card (plug-in unit)
System status panel slot 1 (plug-in unit)
CPU temperature (equipment)
CPU temperature (equipment)
Parallel port 1 instance (termination point)
Ethernet port 2 instances (termination point)
SCSI slot 1 instance (equipment holder)
Serial port 2 instances (termination point)
SCSI slot (equipment holder)
PMC slot (equipment holder)
Ethernet port (termination point)
Drive expansion box (equipment holder)
Drive expansion box (equipment holder)
PMC card (plug-in unit)
Expansion box disk drive 3 instances (equipment)
SCSI slot 1 instance (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)

FIGURE 2-5 Rear-Access Netra CT 810 System View From Host CPU Board

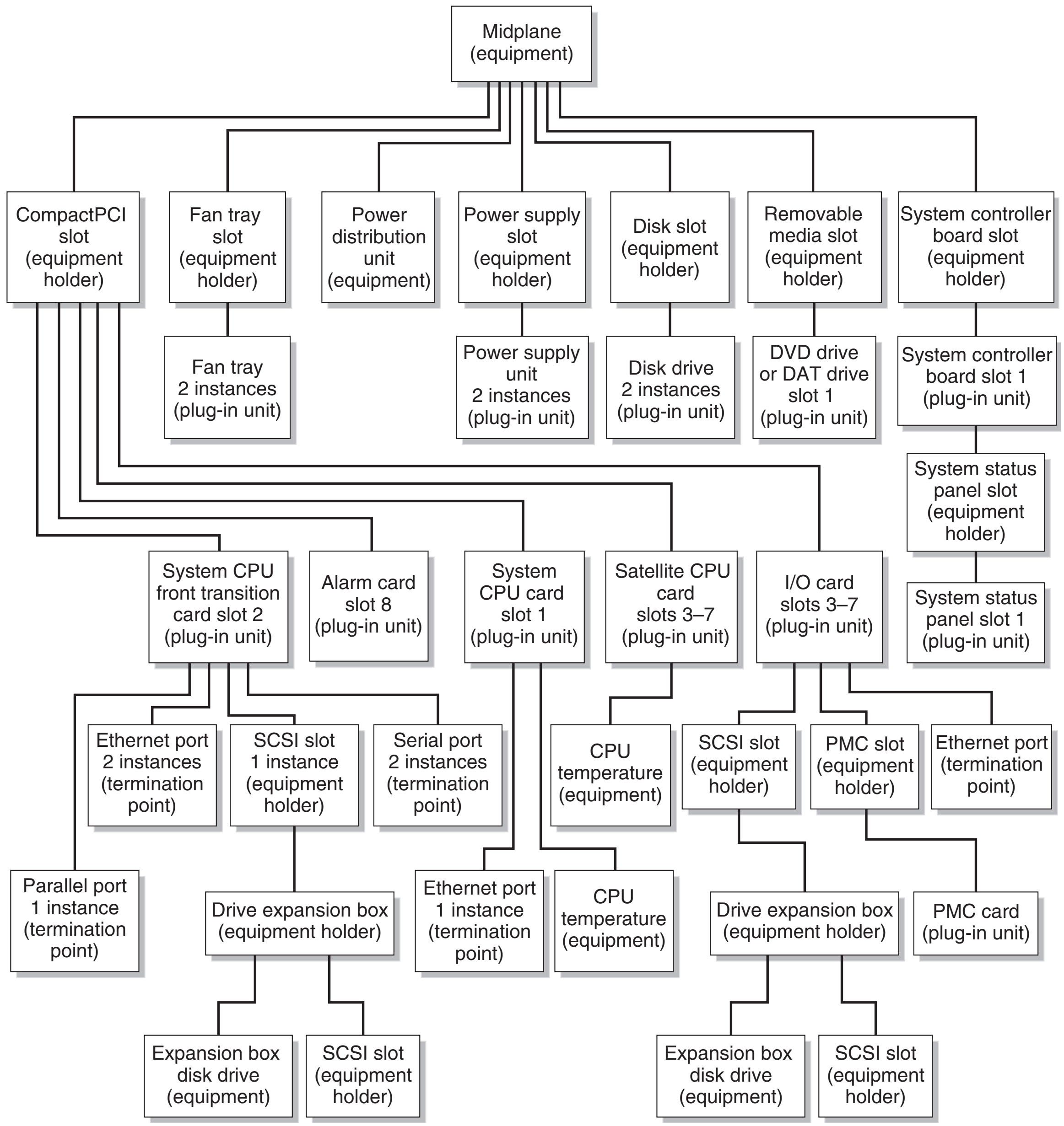

Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Disk slot (equipment holder)
Removable media slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 2 instances (plug-in unit)
Disk drive 2 instances (plug-in unit)
DVD drive or DAT drive slot 1 (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
System status panel 1 (plug-in unit)
System CPU front transition card slot 2 (plug-in unit)
Alarm card slot 8 (plug-in unit)
System CPU card slot 1 (plug-in unit)
Satellite CPU card slots 3–7 (plug-in unit)
I/O card slots 3–7 (plug-in unit)
Ethernet port 2 instances (termination point)
SCSI slot 1 instance (equipment holder)
Serial port 2 instances (termination point)
CPU temperature (equipment)
SCSI slot (equipment holder)
PMC slot (equipment holder)
Ethernet port (termination point)
Parallel port 1 instance (termination point)
Drive expansion box (equipment holder)
Ethernet port 1 instance (termination point)
CPU temperature (equipment)
Drive expansion box (equipment holder)
PMC card (plug-in unit)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)

FIGURE 2-6 Front-Access Netra CT 810 System View From Host CPU Board

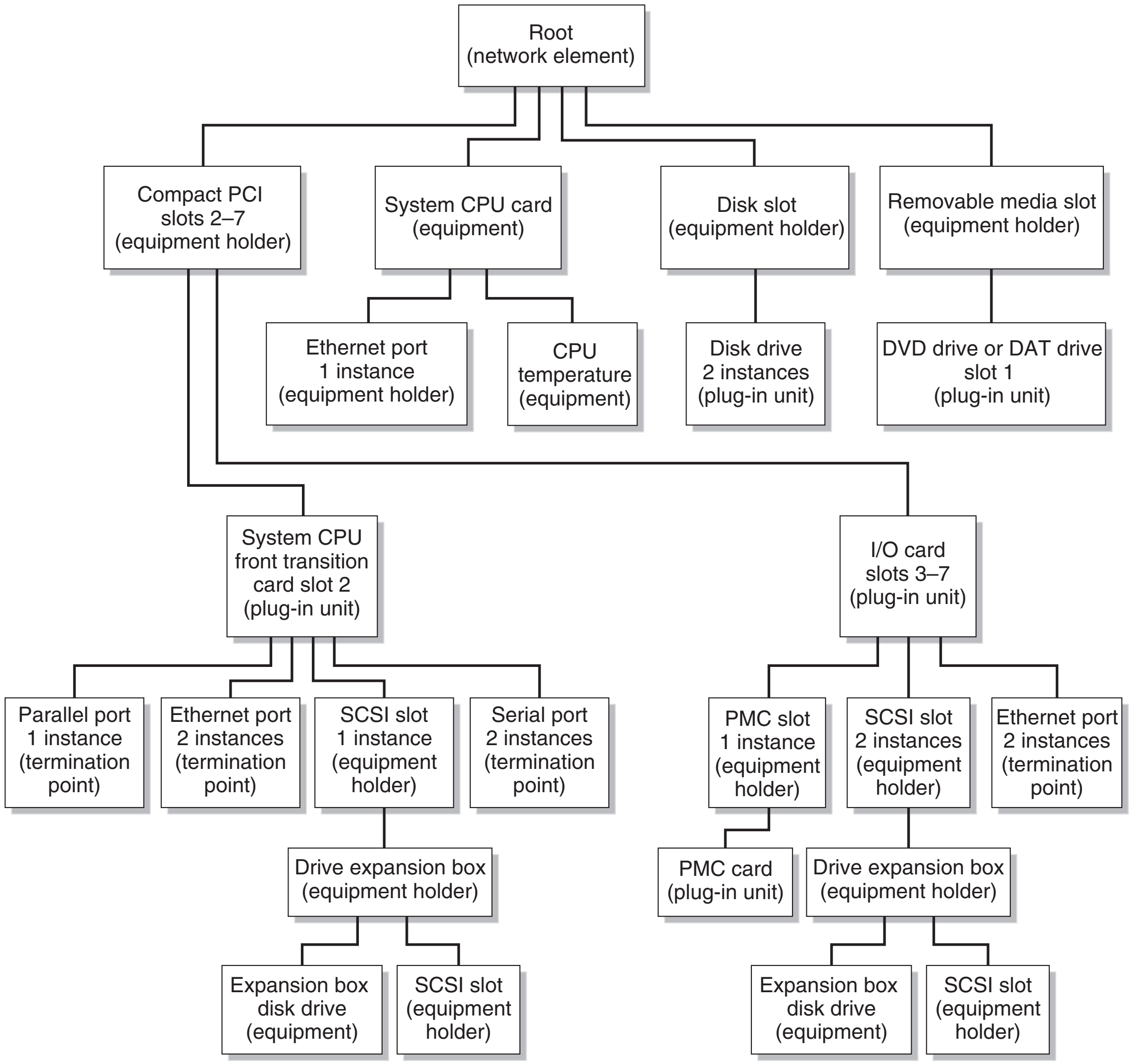


FIGURE 2-7 Front-Access Netra CT 810 System Host CPU Board Local View

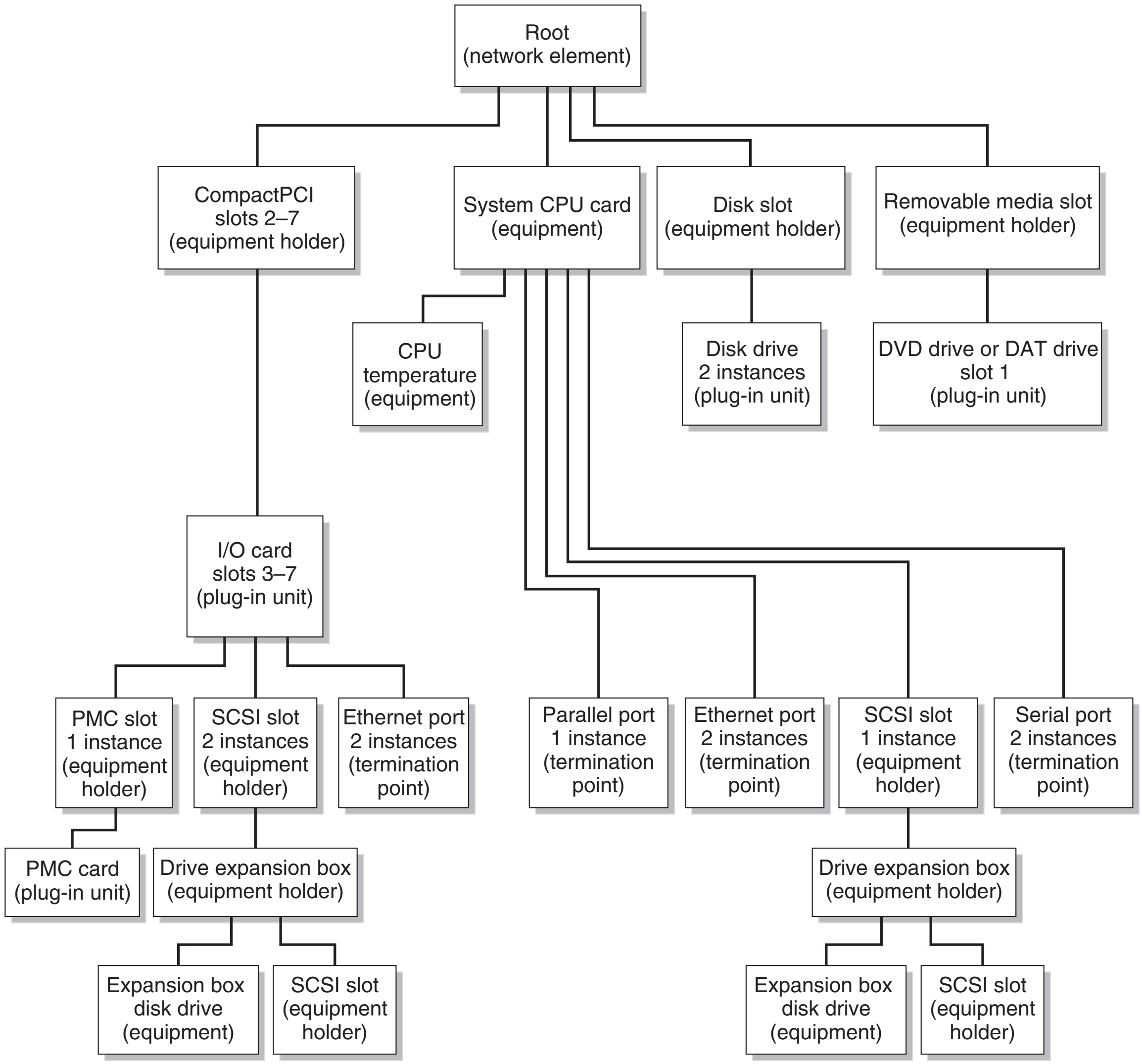

Root (network element)
CompactPCI slots 2–7 (equipment holder)
System CPU card (equipment)
Disk slot (equipment holder)
Removable media slot (equipment holder)
CPU temperature (equipment)
Disk drive 2 instances (plug-in unit)
DVD drive or DAT drive slot 1 (plug-in unit)
I/O card slots 3–7 (plug-in unit)
PMC slot 1 instance (equipment holder)
SCSI slot 2 instances (equipment holder)
Ethernet port 2 instances (termination point)
Parallel port 1 instance (termination point)
Ethernet port 2 instances (termination point)
SCSI slot 1 instance (equipment holder)
Serial port 2 instances (termination point)
PMC card (plug-in unit)
Drive expansion box (equipment holder)
Drive expansion box (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)

FIGURE 2-8 Rear-Access Netra CT 810 System Host CPU Board Local View

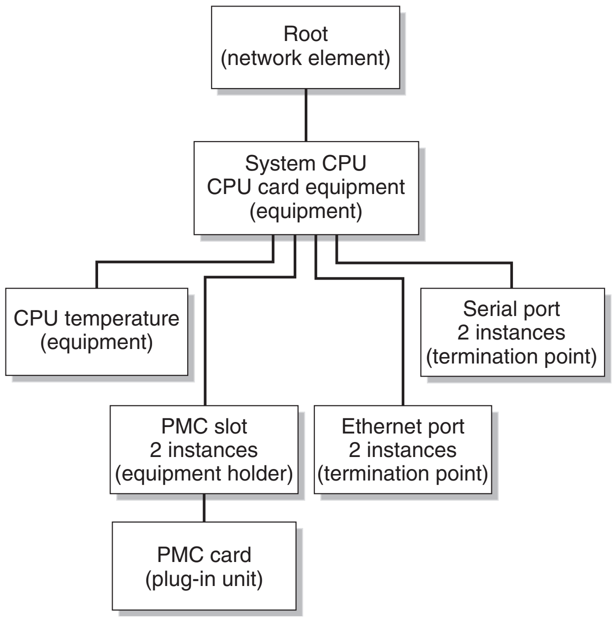


FIGURE 2-9 Rear-Access Netra CT 810 System Satellite CPU Board Local View

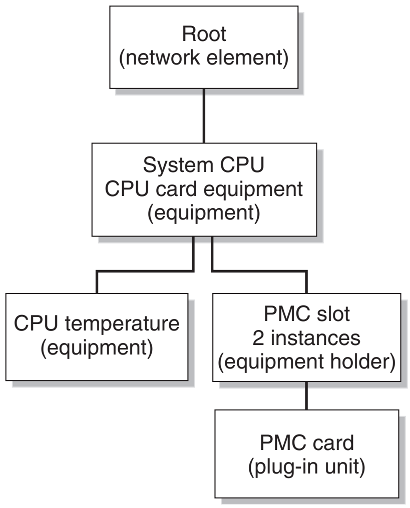


FIGURE 2-10 Netra CT 810 System Satellite CPU Board Local View

Netra CT 410 System Equipment Models

This section contains the following equipment models for the Netra CT 410 server:

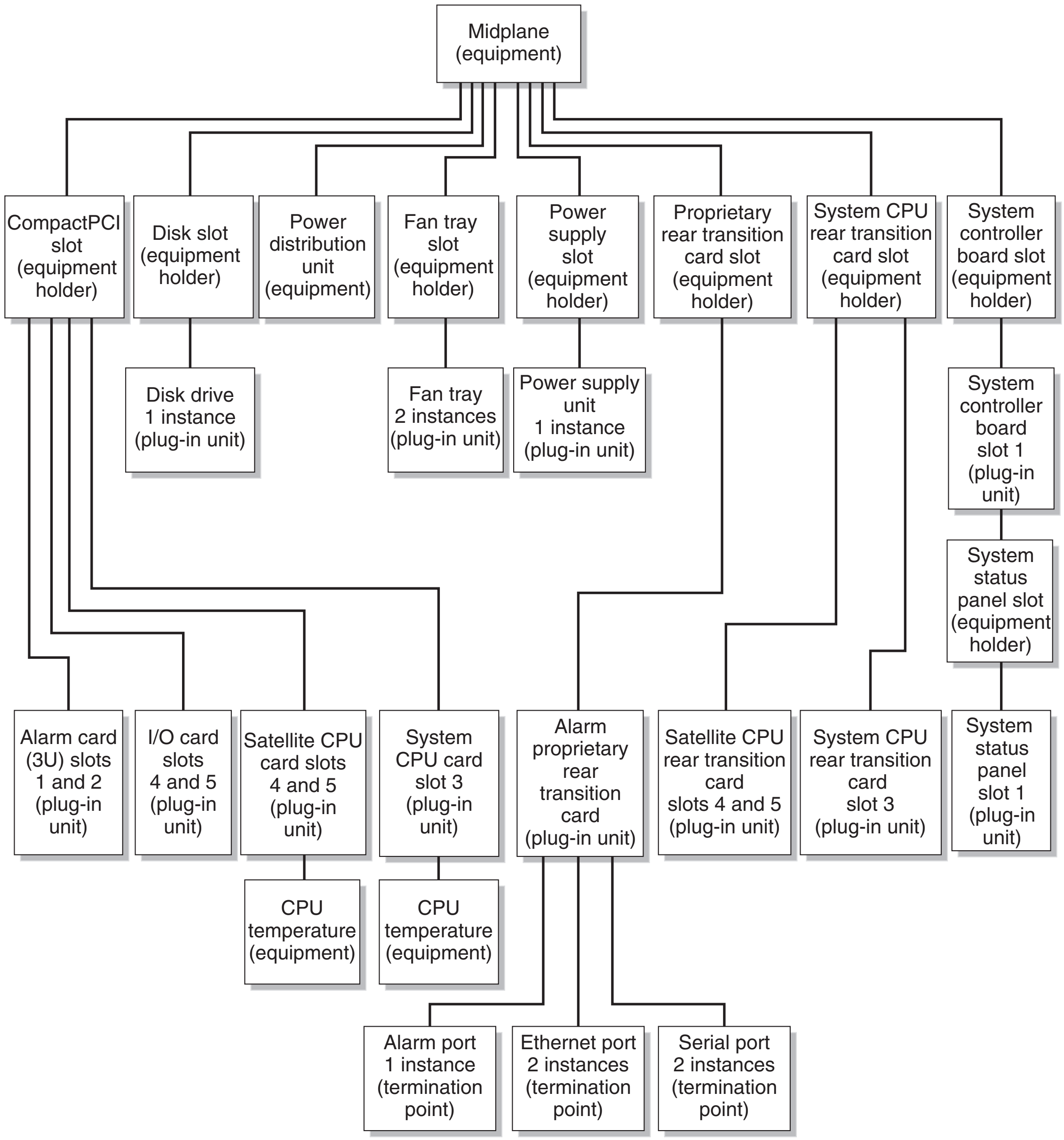
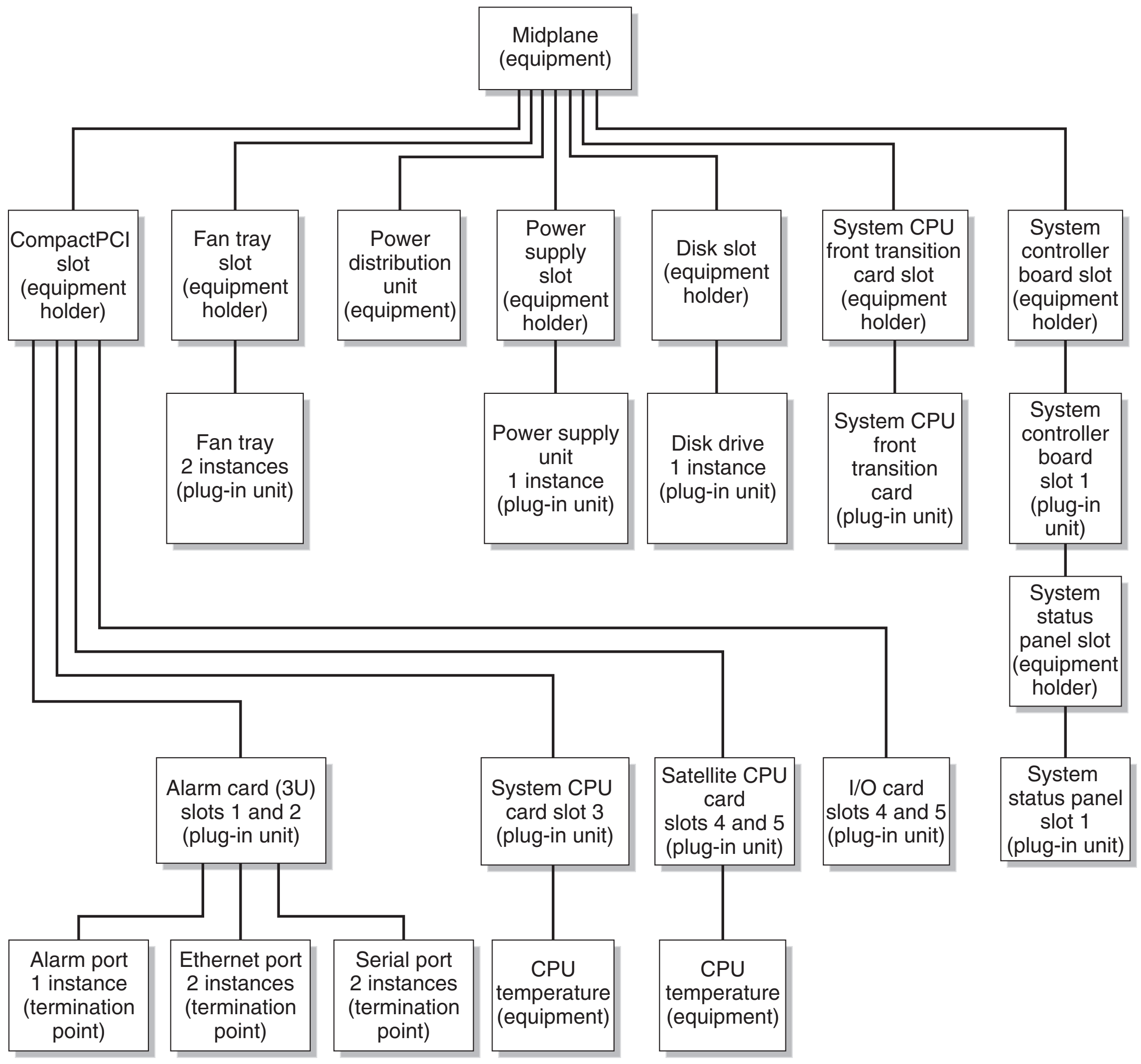

Midplane (equipment)
CompactPCI slot (equipment holder)
Disk slot (equipment holder)
Power distribution unit (equipment)
Fan tray slot (equipment holder)
Power supply slot (equipment holder)
Proprietary rear transition card slot (equipment holder)
System CPU rear transition card slot (equipment holder)
System controller board slot (equipment holder)
Disk drive 1 instance (plug-in unit)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card (3U) slots 1 and 2 (plug-in unit)
I/O card slots 4 and 5 (plug-in unit)
Satellite CPU card slots 4 and 5 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Alarm proprietary rear transition card (plug-in unit)
Satellite CPU rear transition card slots 4 and 5 (plug-in unit)
System CPU rear transition card slot 3 (plug-in unit)
System status panel slot 1 (plug-in unit)
CPU temperature (equipment)
CPU temperature (equipment)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)

FIGURE 2-11 Rear-Access Netra CT 410 Diskfull System View From Alarm Card



Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Disk slot (equipment holder)
System CPU front transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
Disk drive 1 instance (plug-in unit)
System CPU front transition card (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card (3U) slots 1 and 2 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Satellite CPU card slots 4 and 5 (plug-in unit)
I/O card slots 4 and 5 (plug-in unit)
System status panel slot 1 (plug-in unit)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)
CPU temperature (equipment)
CPU temperature (equipment)

FIGURE 2-12 Front-Access Netra CT 410 Diskfull System View From Alarm Card

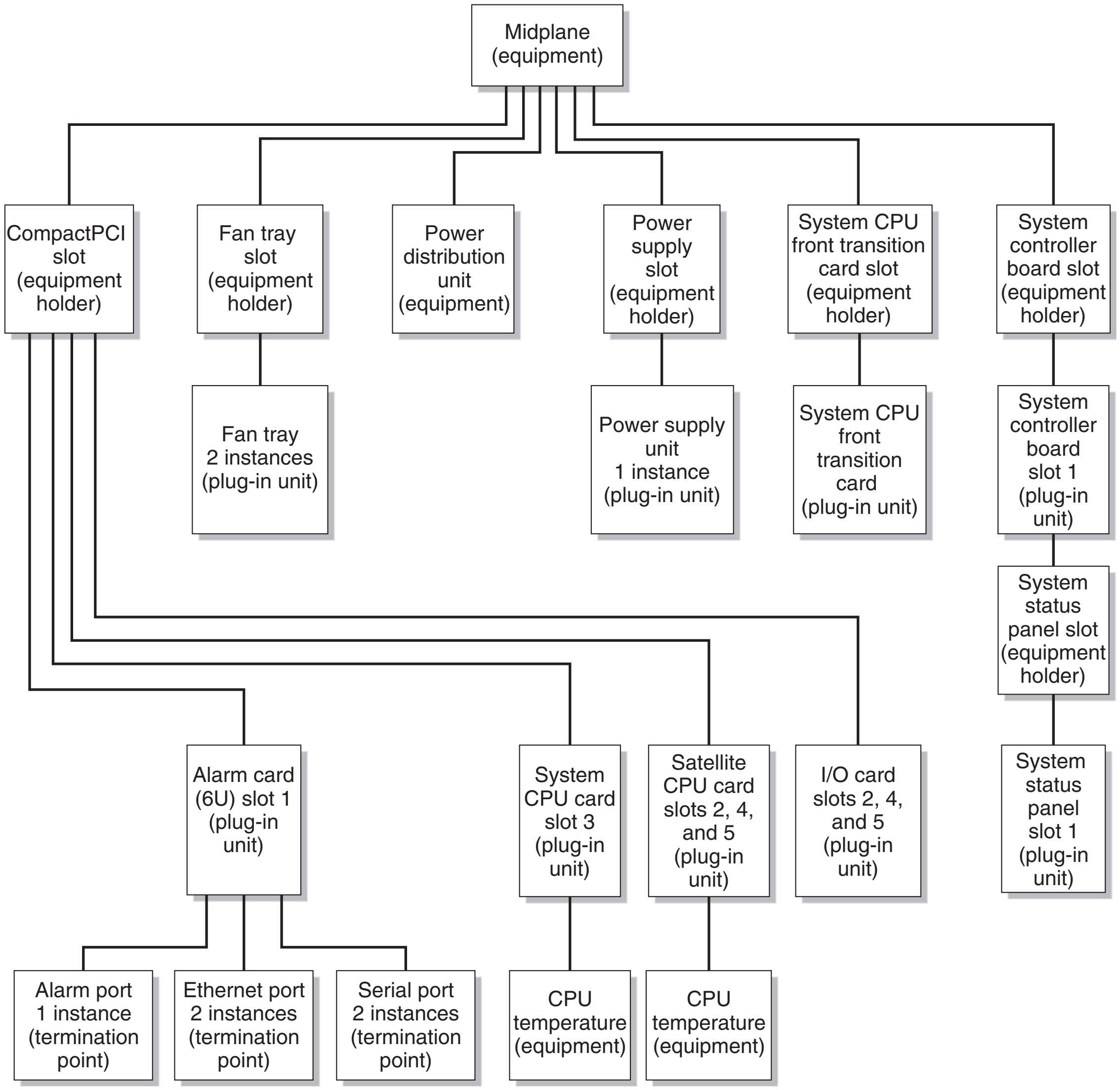
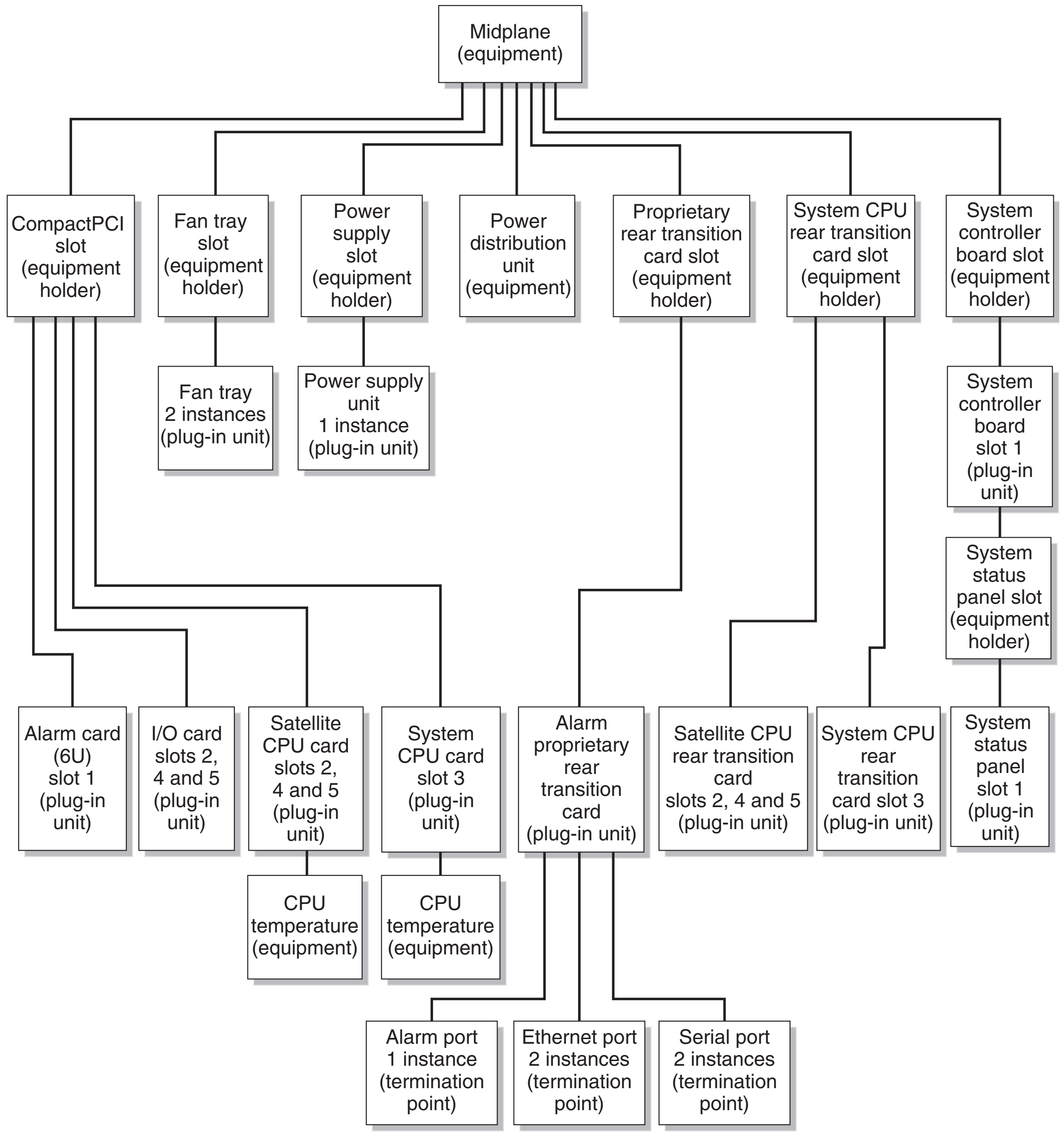

Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
System CPU front transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
System CPU front transition card (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card (6U) slot 1 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Satellite CPU card slots 2, 4, and 5 (plug-in unit)
I/O card slots 2, 4, and 5 (plug-in unit)
System status panel slot 1 (plug-in unit)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)
CPU temperature (equipment)
CPU temperature (equipment)

FIGURE 2-13 Front-Access Netra CT 410 Diskless System View From Alarm Card



Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power supply slot (equipment holder)
Power distribution unit (equipment)
Proprietary rear transition card slot (equipment holder)
System CPU rear transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card (6U) slot 1 (plug-in unit)
I/O card slots 2, 4 and 5 (plug-in unit)
Satellite CPU card slots 2, 4 and 5 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Alarm proprietary rear transition card (plug-in unit)
Satellite CPU rear transition card slots 2, 4 and 5 (plug-in unit)
System CPU rear transition card slot 3 (plug-in unit)
System status panel slot 1 (plug-in unit)
CPU temperature (equipment)
CPU temperature (equipment)
Alarm port 1 instance (termination point)
Ethernet port 2 instances (termination point)
Serial port 2 instances (termination point)

FIGURE 2-14 Rear-Access Netra CT 410 Diskless System View From Alarm Card

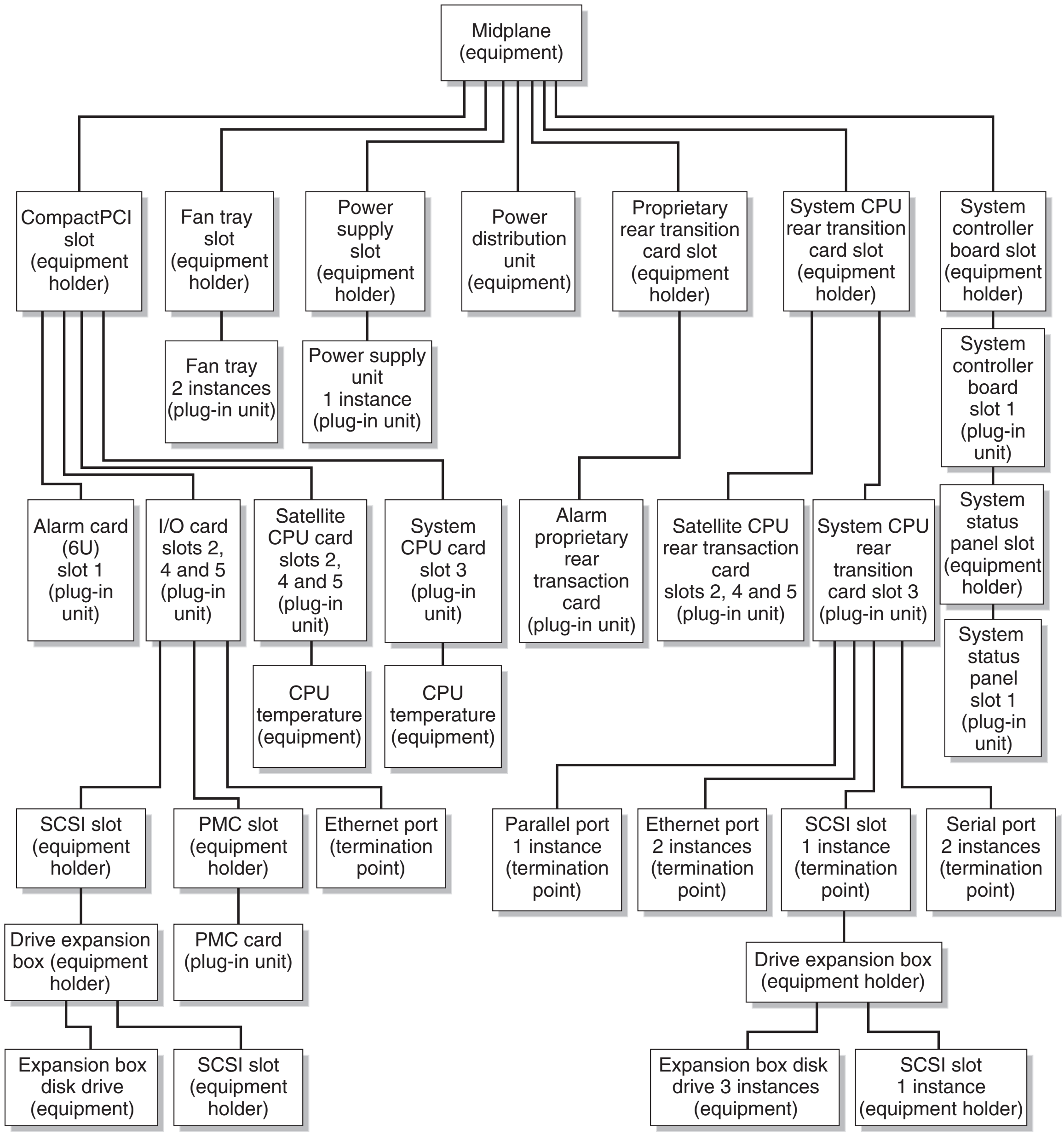
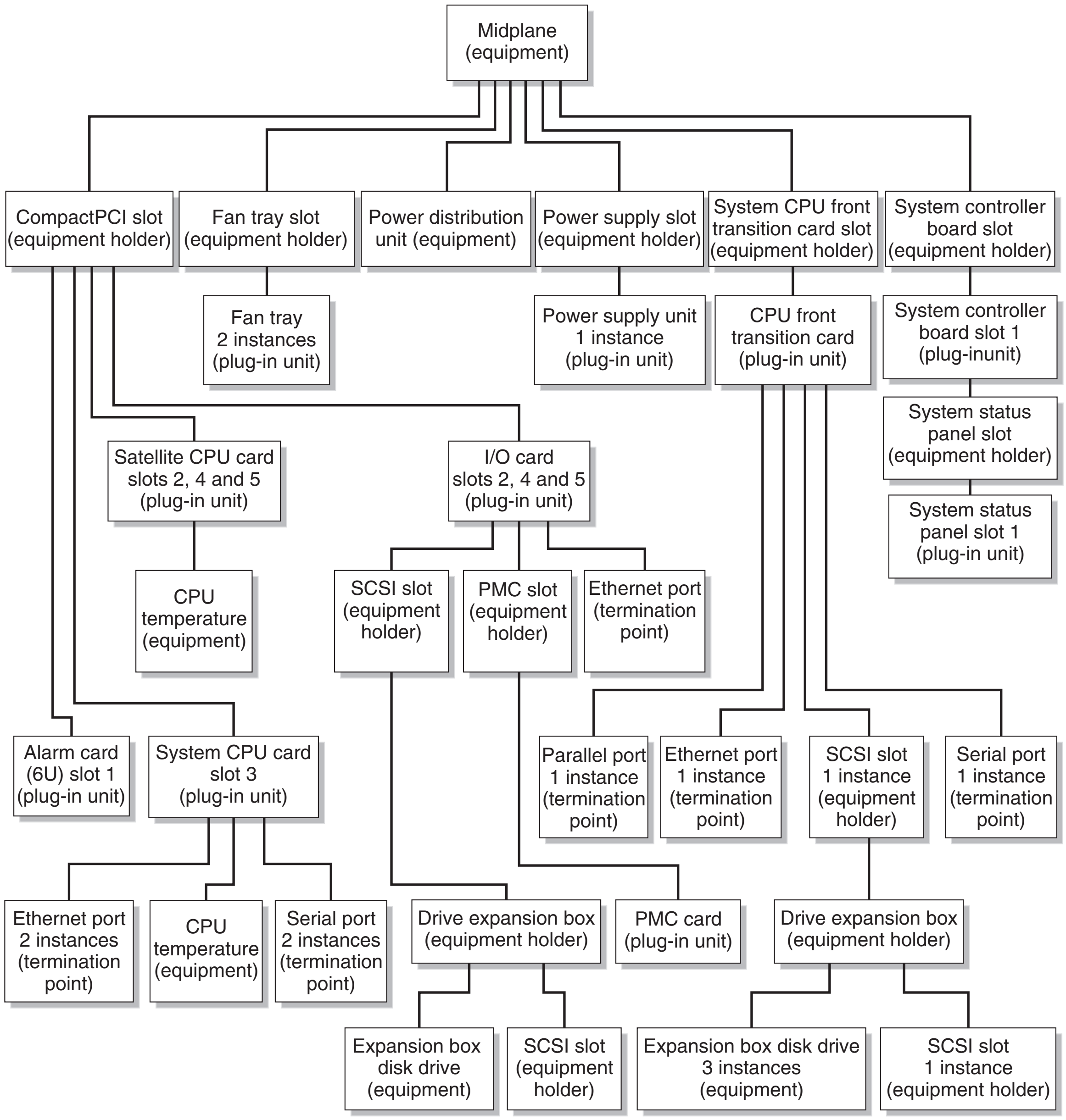

Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power supply slot (equipment holder)
Power distribution unit (equipment)
Proprietary rear transition card slot (equipment holder)
System CPU rear transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
System controller board slot 1 (plug-in unit)
Alarm card (6U) slot 1 (plug-in unit)
I/O card slots 2, 4 and 5 (plug-in unit)
Satellite CPU card slots 2, 4 and 5 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Alarm proprietary rear transaction card (plug-in unit)
Satellite CPU rear transaction card slots 2, 4 and 5 (plug-in unit)
System CPU rear transition card slot 3 (plug-in unit)
System status panel slot (equipment holder)
CPU temperature (equipment)
CPU temperature (equipment)
System status panel slot 1 (plug-in unit)
SCSI slot (equipment holder)
PMC slot (equipment holder)
Ethernet port (termination point)
Parallel port 1 instance (termination point)
Ethernet port 2 instances (termination point)
SCSI slot 1 instance (termination point)
Serial port 2 instances (termination point)
Drive expansion box (equipment holder)
PMC card (plug-in unit)
Drive expansion box (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)
Expansion box disk drive 3 instances (equipment)
SCSI slot 1 instance (equipment holder)

FIGURE 2-15 Rear-Access Netra CT 410 Diskless System View From Host CPU Board



Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
System CPU front transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
CPU front transition card (plug-in unit)
System controller board slot 1 (plug-inunit)
System status panel slot (equipment holder)
System status panel slot 1 (plug-in unit)
Satellite CPU card slots 2, 4 and 5 (plug-in unit)
I/O card slots 2, 4 and 5 (plug-in unit)
CPU temperature (equipment)
SCSI slot (equipment holder)
PMC slot (equipment holder)
Ethernet port (termination point)
Alarm card (6U) slot 1 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Parallel port 1 instance (termination point)
Ethernet port 1 instance (termination point)
SCSI slot 1 instance (equipment holder)
Serial port 1 instance (termination point)
Ethernet port 2 instances (termination point)
CPU temperature (equipment)
Serial port 2 instances (termination point)
Drive expansion box (equipment holder)
PMC card (plug-in unit)
Drive expansion box (equipment holder)
Expansion box disk drive (equipment)
SCSI slot (equipment holder)
Expansion box disk drive 3 instances (equipment)
SCSI slot 1 instance (equipment holder)

FIGURE 2-16 Front-Access Netra CT 410 Diskless System View From Host CPU Board

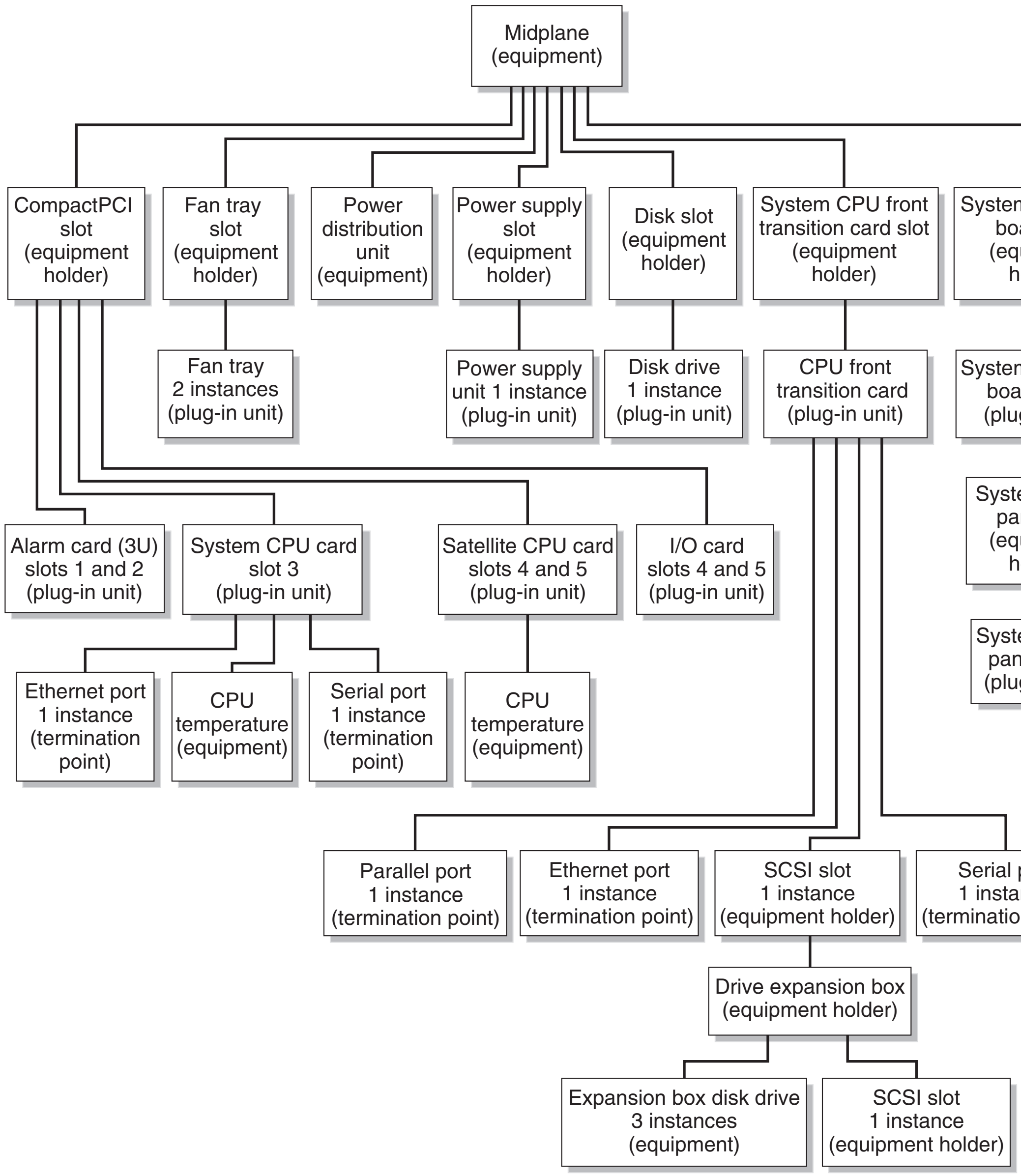

Midplane (equipment)
CompactPCI slot (equipment holder)
Fan tray slot (equipment holder)
Power distribution unit (equipment)
Power supply slot (equipment holder)
Disk slot (equipment holder)
System CPU front transition card slot (equipment holder)
System controller board slot (equipment holder)
Fan tray 2 instances (plug-in unit)
Power supply unit 1 instance (plug-in unit)
Disk drive 1 instance (plug-in unit)
CPU front transition card (plug-in unit)
System controller board slot 1 (plug-in unit)
System status panel slot (equipment holder)
Alarm card (3U) slots 1 and 2 (plug-in unit)
System CPU card slot 3 (plug-in unit)
Satellite CPU card slots 4 and 5 (plug-in unit)
I/O card slots 4 and 5 (plug-in unit)
System status panel slot 1 (plug-in unit)
Ethernet port 1 instance (termination point)
CPU temperature (equipment)
Serial port 1 instance (termination point)
CPU temperature (equipment)
Parallel port 1 instance (termination point)
Ethernet port 1 instance (termination point)
SCSI slot 1 instance (equipment holder)
Serial port 1 instance (termination point)
Drive expansion box (equipment holder)
Expansion box disk drive 3 instances (equipment)
SCSI slot 1 instance (equipment holder)

FIGURE 2-17 Front-Access Netra CT 410 Diskfull System View From Host CPU Board

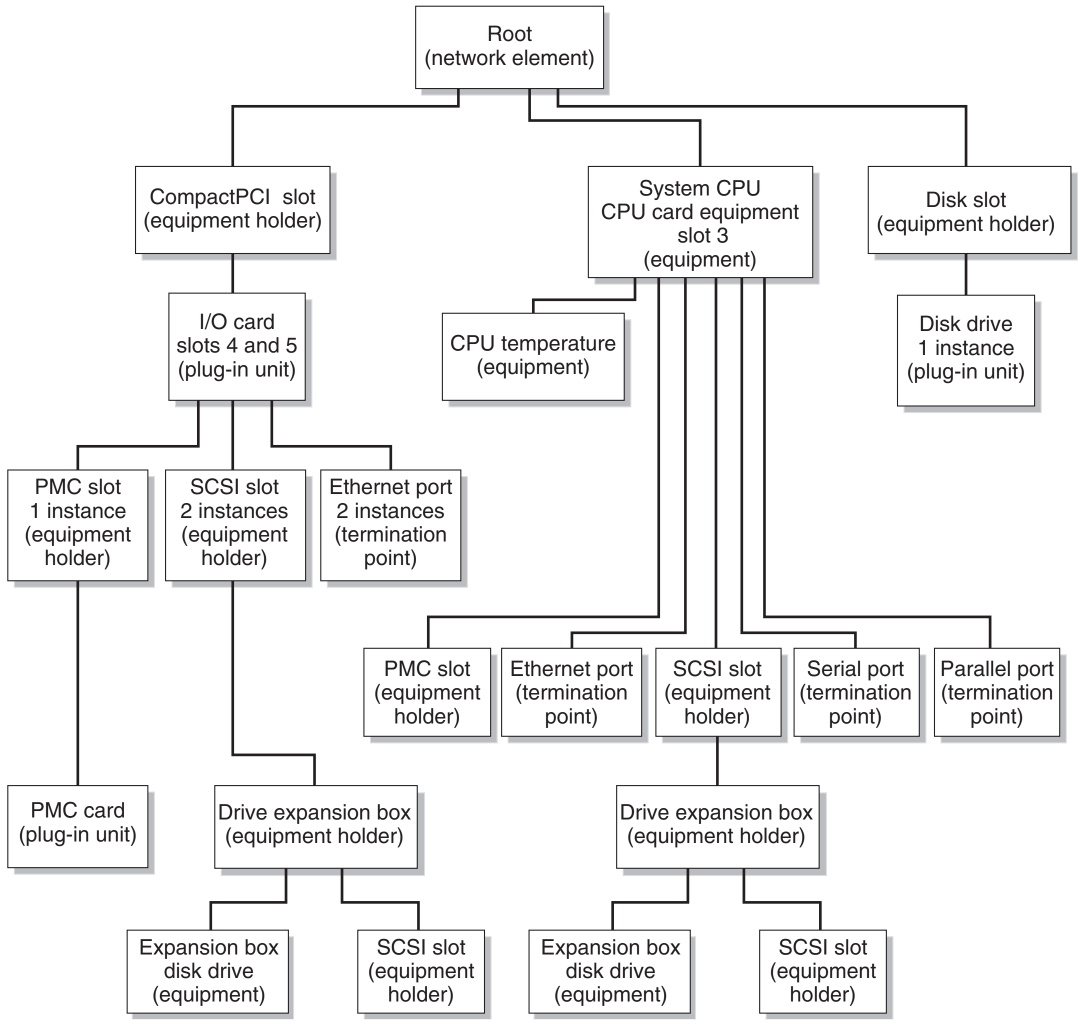


FIGURE 2-18 Rear-Access Netra CT 410 Diskfull Local View From Host CPU Board

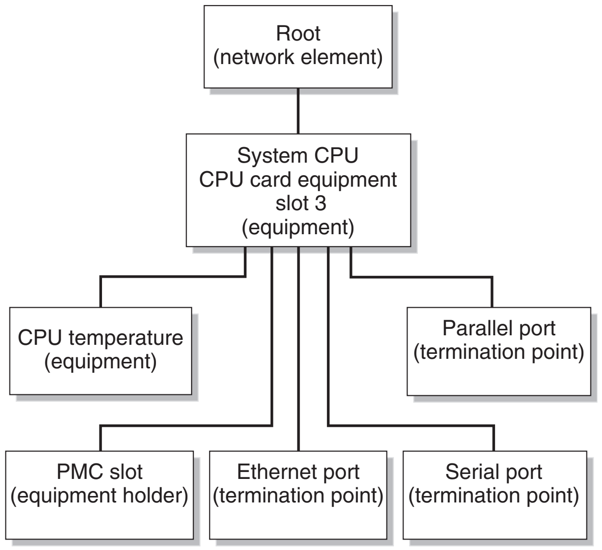


FIGURE 2-19 Rear-Access Netra CT 410 System Satellite CPU Board Local View

CHAPTER 3

Getting Started With Netra CT Element Management Agent API

This chapter explains how to get started writing applications that interface with the Netra CT element management agent, using the Java Management Extensions (JMX) compatible Java API supported by the Netra CT management agent. The chapter consists of:

Before You Begin

You should become acquainted with the topology of the Netra CT server (see Chapter 2), and have some knowledge of Java programming, JMX specifications, and JDMK framework. For more information about JDMK refer to *Java Dynamic Management Kit 4.2 Tutorial* (806-6633), or go to http://`java.sun.com/docs/books/tutorial/index.html`.

Verify that you have the Solaris OS on your development system. In addition, you can download the required Netra CT patch packages from:

`http://sunsolve.com`

These packages consist of:

TABLE 3-1 Solaris Packages for Netra CT Developer APIs

Package	**Description**
`SUNW2jdrt`	Java Runtime Java Dynamic Management Kit (JDMK) package
`SUNWctmgx`	Netra CT management agent package
`SUNWctac`	Alarm card firmware package that includes the Netra CT management agent

You will use these installed packages to work with this tutorial.

About Netra CT Element Management Agent API

The Netra CT server software package includes various modules and extensions (see "Operating System Specifics" on page 4), and the `netract` agent is one of these.

The `netract` agent, when appropriately invoked, provides configuration monitoring and fault monitoring. This enables you to investigate the installed system, and to determine whether the components are running smoothly.

Individual `netract` agents run on the alarm card, the host CPU board, and any satellite CPU board. A management application must be able to talk to the different agents and gather information about the system into a database.

Each `netract` agent notifies the management application of any changes, such as hardware or software configuration changes, and also detects faults when they occur.

The `netract` agent provides two different interfaces for the management applications, one is SNMP version 2C interface, the other is a JMX-compatible Java API called Netra CT management agent API. This chapter provides an introduction on how to write a management application using this Java API.

Netra CT Agent Security

For JMX/JDMK/RMI connectivity, the Netra CT agent provides security by authenticating the application connecting to it via the context of a valid username and password pair.

The username and password must be previously created in the Alarm Card database via Alarm Card CLI. An account on Alarm card consists of username, password and permission. For Netra CT agent, there are only two permissions: read-only and read-write. User account on Alarm Card must have ALL PRIVILEDGES ENABLED to have read-write permission. (See the *Netra CT Server System Administration Guide* for details on setting up user accounts.)

There is a *security flag* used to enable and disable the Security Feature. This flag is stored persistently and its default value is false. The security flag can be set to true or false via Alarm Card CLI command `setmohsecurity`. A reset of Alarm Card is required after changing the flag for the feature to take effect. (See the *Netra CT Server System Administration Guide* for information on the `setmohsecurity` and `showmohsecurity` CLI commands).

You can get the state of security flag with the Alarm Card CLI command `showmohsecurity` or with the API by using the NEMBean's `getSecurityFlag` method described in "NEMBean" on page 95.

If the flag is **true**, security is on. This means the application that connects to Netra CT agent must provide a valid username and password to be able to establish connection.

If the flag is **false**, security is off and no authentication is done. It does not matter whether an application provides username, password or not, it is always allowed to connect.

Sample code with Netra CT Security is shown CODE EXAMPLE 3-1.

CODE EXAMPLE 3-1 Sample code with Netra CT Security

```
...
// set up the authentication info
AuthClient.setAuthInfo(connectorClient, username, password);
// now connect to the agent...
connectorClient.connect();
...
```

Creating Your Application

Creating an application to interface with and manage the configuration of the Netra CT server involves a series of steps. You must be able to:

- Enquire into the hierarchy of the system configuration
- Monitor notifications
- Monitor alarms

1. **Cut and paste the relevant code example into a text editor, make any necessary adjustments, and compile the code.**

   Make sure that `SUNW2jdtk` is installed before trying to compile `Client.java`. Refer to the *Java Dynamic Management Kit 4.2 Tutorial* for background information on `Client.java`.

2. **To compile** `Client.java`**, issue the command** `/usr/j2se/bin/javac -classpath`:

```
$ /usr/j2se/bin/javac -classpath \
/opt/SUNWjdmk/jdmk4.2/1.2/lib/jdmkrt.jar: \
/opt/SUNWnetract/mgmt2.0/lib/agent.jar Client.java
```

   Compiling `Client.java` should produce the file `Client.class`. If you have difficulty, refer to the Java Tutorial example of running a simple client.

3. **Before running** `Client.java`**, start the agent by issuing the command** `/opt/SUNWnetract/mgmt2.0/bin/ctmgx start`

```
$ /opt/SUNWnetract/mgmt2.0/bin/ctmgx start
```

4. **Use the following command to run** `Client.java`:

```
$ /usr/j2se/bin/java -classpath \
.:/opt/SUNWjdmk/jdmk4.2/1.2/lib/jdmkrt.jar: \
/opt/SUNWnetract/mgmt2.0/lib/agent.jar Client
```

   The following sections point out various features of the Netra CT element management API.

Purpose of the Application

First, a management application needs to know how the system is configured. The simplest example sets up an agent describing the hardware containment hierarchy. From the root of this tree, the management tree can be developed to show, for example, how many fans there are, which cards are in which slots and so on. Developing code that begins this action is the purpose of "Determining the System Configuration Hierarchy" on page 35.

"Listening for Notifications" on page 38 deals with developing a way of monitoring notifications such as power on and power off to a particular slot or device.

"Managing Alarms" on page 39 covers alarm management: how to handle the receiving and transmitting of system alarms such as CPU over-temperature alarm.

"Software Monitoring" on page 46 shows how to get a list of running software services and daemons, so they can be registered to receive notice of events.

Determining the System Configuration Hierarchy

In this section you develop a client to print out the object names of the MBeans representing the system. A complete description of Mbeans, together with examples, can be found in the JDMK documentation.

1. **Ensure you have the appropriate software installed on the development system for the application you intend to develop.**

   Refer to the *Netra CT Server System Administration Guide* if you need help in installing the appropriate software.

2. **Go to:** `/opt/SUNWnetract/mgmt2.0/docs/api` **to find the documentation that identifies the pieces you need to communicate with the** `netract` **agent.**

   See the API documentation for:

   - `com.sun.ctmgx.MohNames`
   - `com.sun.ctmgx.ContainmentTreeMbean`

   For JDMK documentation, go to: `/opt/SUNWjdmk/jdmk4.2/1.2/docs`. For an introduction to JDMK, go to `//docs.sun.com`, and search for the Java Dynamic Management Kit 4.2 Tutorial.

   See the API documentation for:

   - `com.sun.jdmk.comm.RmiConnectorAddress`
   - `com.sun.jdmk.comm.RmiConnectorClient`

Communicating to the Netra CT Agent

This simple demonstration lets you connect a client with an instance of `netract` agent, beginning in CODE EXAMPLE 3-2. This example represents part one of a three-part example. A detailed explanation follows.

CODE EXAMPLE 3-2 Creating a Client to Communicate With the Netra CT Agent (Part 1)

```
import java.util.Iterator;
import java.util.Set;
import javax.management.ObjectName;
import com.sun.ctmgx.moh.MohNames;
```

CODE EXAMPLE 3-2 Creating a Client to Communicate With the Netra CT Agent (Part 1)

```
import com.sun.jdmk.ServiceName;
import com.sun.jdmk.comm.RmiConnectorAddress;
import com.sun.jdmk.comm.RmiConnectorClient;

public class Client {

    private RmiConnectorClient connectorClient;
    private RmiConnectorAddress connectorAddress;

    public Client() {
        connectorClient = new RmiConnectorClient();
        connectorAddress = new RmiConnectorAddress();
    }

    public static void main(String[] args) {
        Client client = new Client();
        try {
            client.printContainmentTree();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

CODE EXAMPLE 3-2 instantiates the **RmiConnectorClient** and **RmiConnectorAddress**.

The demonstration continues in CODE EXAMPLE 3-3.

Finding the Root Object Name

CODE EXAMPLE 3-3 continues from the previous example. The code example connects to the client and prints the ContainmentTree by getting the ObjectName of the root MBean in the containment hierarchy.

Each **MohNames** instance comes up with **ObjectNames** instances that are accessible via public static fields defined in MohNames. This includes the **ContainmentTreeMBean** instance, which provides a mechanism for the user to traverse the containment hierarchy representing the Netra CT system.

CODE EXAMPLE 3-3 Getting the Root MBean Object Name (Part 2)

```
  public void printContainmentTree() throws Exception {
        connectorClient.connect(connectorAddress);

        Object[] params = new Object[0];
        String[] signature = new String[0];
```

CODE EXAMPLE 3-3 Getting the Root MBean Object Name (Part 2)

```
        ObjectName rootName =
            (ObjectName)connectorClient.invoke(MohNames.MOH_CONTAINMENT_TREE,
                                                 "getRoot", params, signature);
        printSubTree(rootName);

        connectorClient.disconnect();
    }
```

This demonstration returns the object name of the instance of **NEMBean**. NEMBean is the name of the network element MBean representing the system as a whole, in other words, the root of the tree.

Now that you have identified the **ObjectName** of the root of the `MOH_CONTAINMENT_TREE` you are ready to traverse the tree and find out what other elements are in the tree.

Traversing the Containment Hierarchy From a Node

Continuing the demonstration from the previous example, in CODE EXAMPLE 3-4 you traverse the `MOH_CONTAINMENT_TREE` from a node, and can get a list of all the nodes on the tree using `getChildren`.

CODE EXAMPLE 3-4 Traversing the Containment Hierarchy From a Node (Part 3)

```
    private void printSubTree(ObjectName nodeName) throws Exception {
        System.out.println(nodeName);

        Object[] params = {nodeName};
        String[] signature = {"javax.management.ObjectName"};

        Set children =
            (Set)connectorClient.invoke(MohNames.MOH_CONTAINMENT_TREE,
                                         "getChildren", params, signature);

        for (Iterator itr = children.iterator(); itr.hasNext();) {
            printSubTree((ObjectName)itr.next());
        }
    }
}
```

Here, the **nodeName** is the ObjectName of the MBean where the search should start from. The line beginning `Set children` gets the children of the specified MBean in the containment hierarchy.

Once you have established the hierarchy of the existing system, your application must receive notification when changes to the system occur. This is the subject of the following section.

Listening for Notifications

This series of examples assumes you continue from the previous three-part example. Return to `/opt/SUNWnetract/mgmt2.0/docs/api` to find documentation.

In the JDMK framework, look at:

- `javax.management.Notification`
- `javax.management.NotificationListener`
- `javax.management.NotificationFilterSupport`
- `javax.management.NotificationFilter`

In MOH documentation you need to look at: `com.sun.ctmgx.moh.MohNames` and `com.sun.ctmgx.moh.Moh.EFDMBean`.

Registering a Notification Listener With EFDMBean Instance

This example continues from the previous examples and shows you how to register a `NotificationListener` using a `NotificationFilter`.You begin by adding a NotificationListener that catches communications from the RmiConnectorClient.

CODE EXAMPLE 3-5 RMI Example of Listening for MOH Notifications

```
Registering a NotificationListener with a NotificationFilter

      try {
            // accessing MohNames for MOH_DEFAULT_EFD
            //
            connectorClient.addNotificationListener(MohNames.MOH_DEFAULT_EFD,\
                                                    aListener, aFilter, null);
       }
      catch (com.sun.jdmk.comm.CommunicationException ce) {
            try {
                 connectorClient.setMode(RmiConnectorClient.PULL_MODE);
                 connectorClient.addNotificationListener\
                    (MohNames.MOH_DEFAULT_EFD,aListener, aFilter, null);
            }
            catch (Exception e) {
            }
       }
```

CODE EXAMPLE 3-5 establishes that **MohNames** can access MOH_DEFAULT_EFD. The **EFDMBean** exposes the remote management interface of an event forwarding discriminator managed object.

The netract agent of the Netra CT alarm card does not support the PUSH_MODE, so the above code will work for any of the netract agent instances (those on the host, satellite, and alarm card) in a Netra CT drawer.

Managing Alarms

Before you begin this segment of code example, you should refer back to: /opt/SUNWnetract/mgmt2.0/docs/api

Look at the MOH documentation for:

- com.sun.ctmgx.moh.AlarmNotification
- com.sun.ctmgx.moh.AlarmNotificationFilter
- com.sun.ctmgx.moh.AlarmSeverity
- com.sun.ctmgx.moh.AlarmSeverityProfileMBean
- com.sun.ctmgx.moh.AlarmType

Registering a Notification Listener With an Alarm Notification Filter

In this section you identify the kinds of alarms the script listens for when events occur. You can specify the level of action; this example listens for critical or major alarms. **AlarmNotification** represents an alarm notification emitted by an MBean.

CODE EXAMPLE 3-6 Registering a NotificationListener With an AlarmNotificationFilter

```
AlarmNotificationFilter aFilter = new AlarmNotificationFilter();

// interested in all types of alarms
//
aFilter.enableAllAlarmTypes();

// interested in only CRITICAL and MAJOR alarms
//
aFilter.enableSeverity(AlarmSeverity.CRITICAL);
aFilter.enableSeverity(AlarmSeverity.MAJOR);

try {
            connectorClient.addNotificationListener(MohNames.MOH_DEFAULT_EFD,\
                                                        aListener, aFilter, null)
            }
```

CODE EXAMPLE 3-6 Registering a NotificationListener With an AlarmNotificationFilter

```
catch (com.sun.jdmk.comm.CommunicationException ce) {
            connectorClient.setMode(RmiConnectorClient.PULL_MODE);
            connectorClient.addNotificationListener(MohNames.MOH_DEFAULT_EFD,\
                                                    aListener, aFilter, null)
            }
            catch (Exception e) {
            }
```

CODE EXAMPLE 3-6 follows the form of the previous example in setting the **RmiConnectorClient** to `PULL_MODE`. The alarm filter is set to **enableAllAlarmTypes**, then refined to enable only `AlarmSeverity.CRITICAL` and `AlarmSeverity.MAJOR`.

Using the Default Alarm Severity Profile

Each `netract` agent instance comes up with a default instance of **AlarmSeverityProfile** which can be accessed by its object name, `MohNames.MOH_DEFAULT_ASP`. The MBean instances that might generate `AlarmNotifications` will have this default alarm severity profile associated with them. The user can associate a new profile to any of those MBeans at any time.

CODE EXAMPLE 3-7 Using the Default Alarm Severity Profile

```
// Get the alarm severity association of the default profile
//
Object[] allObjs = null;
Object obj = null;
Java.util.Set mySet = null;
Java.util.Map myMap = null;
        try {
            myMap = (Map)connectorClient.invoke(MohNames.MOH_DEFAULT_ASP,\
                      "getAlarmSeverityList", null,null);

            mySet = (Set)myMap.keySet();
            allObjs = mySet.toArray();
        } catch(Exception e) {
            e.printStackTrace();
        }

        AlarmType aType = null;
        AlarmSeverity aSeverity = null;

        for (int i = 0; i < mySet.size();i++) {
            try {
                // aType and aSeverity is the association in this
```

CODE EXAMPLE 3-7 Using the Default Alarm Severity Profile *(Continued)*

```
                 // default profile
                 aType = (AlarmType)allObjs[i];
                 aSeverity = (AlarmSeverity)myMap.get(aType);

                 // setting the severity of high temp alarm to critical
                 //
                 if (aType.equals(AlarmType.HIGH_TEMPERATURE)) {
                     Object[] params = new Object[2];
                     String[] signature = new String[2];
                     params[0] = aType;
                     params[1] = AlarmSeverity.CRITICAL;
                     signature[0] = "com.sun.ctmgx.moh.AlarmType";
                     signature[1] = "com.sun.ctmgx.moh.AlarmSeverity";
                     connectorClient.invoke(MohNames.MOH_DEFAULT_ASP, \
                           "setAlarmSeverity", params, signature);


             } catch(Exception e) {
                 e.printStackTrace();
             }
         }
```

In CODE EXAMPLE 3-7, the severity level of `HIGH.TEMPERATURE` AlarmType in the default alarm severity profile has been set to `CRITICAL`. The following example shows how to create your own alarm severity profile instances.

Creating Your Own Alarm Severity Profile

You can create your own alarm severity profile by following CODE EXAMPLE 3-8.

CODE EXAMPLE 3-8 Creating an Alarm Severity Profile

```
try {
    // You need to provide the class name to instantiate an MBean,
    // for AlarmSeverityProfileMBean
    // the class name string is defined by the constant MohNames.CLASS_NAME_ASP
    //
            ObjectName profileName = new ObjectName("NetraCT:name=\
                  AlarmSeverityProfile,id=2");
            connectorClient.createMBean(MohNames.CLASS_NAME_ASP, profileName,\
                  null,null);

   // To make the profile usable, you need to provide the alarm type and severity
   // associations
   //
```

CODE EXAMPLE 3-8 Creating an Alarm Severity Profile *(Continued)*

```
            Object[] params = new Object[2];
            String[] signature = new String[2];
            signature[0] = "com.sun.ctmgx.moh.AlarmType";
            signature[1] = "com.sun.ctmgx.moh.AlarmSeverity";

            // For high temperature alarm
            //
            params[0] = AlarmType.HIGH_TEMPERATURE;
            params[1] = AlarmSeverity.CRITICAL;
            connectorClient.invoke(profileName, \
                   "setAlarmSeverity", params, signature);

            // For high memory utilization alarm
            //
            params[0] = AlarmType.HIGH_MEMORY_UTILIZATION;
            params[1] = AlarmSeverity.MAJOR;
            connectorClient.invoke(profileName,\
                   "setAlarmSeverity", params, signature);

          // For fan failure alarm (NetraCT agent does not support this alarm
          // currently
          //
            params[0] = AlarmType.FAN_FAILURE;
            params[1] = AlarmSeverity.MINOR;
            connectorClient.invoke(profileName,\
                   "setAlarmSeverity", params, signature);

          // For fuse failure alarm (NetraCT agent does not support this alarm
          // currently
          //
            params[0] = AlarmType.FUSE_FAILURE;
            params[1] = AlarmSeverity.WARNING;
            connectorClient.invoke(profileName,\
                   "setAlarmSeverity", params, signature);

        } catch (Exception e) {
            e.printStackTrace();
        }
```

CODE EXAMPLE 3-8 assigns alarm notifications for high memory usage, fan failure, and fuse failure, although the current `netract` agent does not support alarm notifications for fuse failure. The code example is included here for demonstration purposes.

Assigning a New Alarm Severity Profile

CODE EXAMPLE 3-9 shows how to assign a new alarm severity profile to an MBean which can generate AlarmNotifications.

CODE EXAMPLE 3-9 Assigning a New Alarm Severity Profile

```
try {
            Object[] params = new Object[1];
            String[] signature = new String[1];

            signature[0] = "javax.management.ObjectName";

          // pass the object name of the newly created AlarmSeverityProfileMBean
          // instance
          //
            params[0] = profileName;

          // sensorObjectName is the object name of lets say a temperature sensor
          // MBean instance
          //
            connectorClient.invoke(sensorObjectName,\
                "setAlarmSeverityProfilePointer", params, signature);

        } catch (Exception e) {
               e.printStackTrace();
        }
```

The new alarm severity profile can be reserved to replace the default profile when required.

Configuring the Agent to Drive Alarm Card Alarm Outputs

The system configuration hierarchy indicates the physical alarm port which corresponds to a termination point, as shown in "Hardware Resource Hierarchy Showing Managed Object Classes" on page 10 and subsequent views. The alarm port termination point supports five alarm interfaces: three for output, two for input. In general, when an alarm occurs, the corresponding output alarm pin is driven high based on the alarm severity.

The output alarm pins (alarm0, alarm1, alarm2) are statically mapped into severities of critical, major, and minor respectively.

For example, assume that `HIGH_TEMPERATURE` is assigned as critical, and `HIGH_MEMORY_UTILIZATION` is assigned as minor. When a high temperature occurs, alarm0 is driven high to indicate a critical alarm. When a `HIGH_MEMORY_UTILIZATION` occurs, alarm2 is driven high to indicate a minor alarm.

TABLE 3-2 Example of Alarm Output Mapping

alarm0	alarm1	alarm2
critical	major	minor
HIGH_TEMPERATURE		HIGH_MEMORY_UTILIZATION

In JMX, an alarm is defined as a notification with a severity associated with it. These alarms are assigned as **NetworkInterfaceMBeans**, each of which represent a network interface object in the system. Refer to "NetworkInterfaceMBean" on page 98.

You can configure an alarm card agent to drive output alarms from the alarm card on the Netra CT server using MOH as described in the following section.

▼ To Set Up and Use Alarm Features

The following steps show how to configure an agent from the alarm card to correspond with the mapping in TABLE 3-2.

1. **Register a notification listener with an AlarmNotificationFilter.**

   Use the examples beginning "Registering a Notification Listener With an Alarm Notification Filter" on page 39, and modify the default to listen for critical or major alarms. Return to the start of this chapter for help in getting an ObjectName.

2. **Develop an AlarmSeverityProfile based on the default profile.**

   An **AlarmSeverityProfile** (ASP) contains multiple entries, and can be assigned to several alarm-generating objects. Some entries in the profile might not be used by an object, because that object might not be generating that specific kind of alarm. The default instance of **AlarmSeverityProfile** can be accessed by its object name, `MohNames.MOH_DEFAULT_ASP`.

3. **Assign the AlarmSeverityProfile to the corresponding objects.**

   - Assign HIGH_TEMPERATURE to the corresponding CPU thermistor SensorObject.
   - Assign HIGH_MEMORY_UTILIZATION to the corresponding CpucardEquipment object.

In CODE EXAMPLE 3-10 extracted from "Using the Default Alarm Severity Profile" on page 40, the severity level of HIGH.TEMPERATURE AlarmType in the default ASP has been set to CRITICAL corresponding with alarm0.

CODE EXAMPLE 3-10 Extract of Using the Default Alarm Severity Profile

```
// Get the alarm severity association of the default profile
//
<snip>
 .....
                // setting the severity of high temp alarm to critical
                //
                if (aType.equals(AlarmType.HIGH_TEMPERATURE)) {
                    Object[] params = new Object[2];
                    String[] signature = new String[2];
                    params[0] = aType;
                    params[1] = AlarmSeverity.CRITICAL;
                    signature[0] = "com.sun.ctmgx.moh.AlarmType";
                    signature[1] = "com.sun.ctmgx.moh.AlarmSeverity";
                    connectorClient.invoke(MohNames.MOH_DEFAULT_ASP, \
                          "setAlarmSeverity", params, signature);
.....
<unsnip>
```

Any number of objects are capable of generating an alarm. If you assign this profile to a particular object, whenever a hardware failure of that object occurs, the netract agent refers to the profile and responds as you have specified.

CODE EXAMPLE 3-11 creates your own alarm severity profile instances based on these examples. In this case, the **sensorObjectName** is the object name of a temperature sensor MBean instance.

CODE EXAMPLE 3-11 Extract of Assigning a New Alarm Severity Profile

```
try {
            Object[] params = new Object[1];
            String[] signature = new String[1];

            signature[0] = "javax.management.ObjectName";

         // pass the object name of the newly created AlarmSeverityProfileMBean
         // instance
         //
            params[0] = profileName;

         // sensorObjectName is the object name of lets say a temperature sensor
         // MBean instance
```

CODE EXAMPLE 3-11 Extract of Assigning a New Alarm Severity Profile *(Continued)*

```
           //
              connectorClient.invoke(sensorObjectName,\
                  "setAlarmSeverityProfilePointer", params, signature);

          } catch (Exception e) {
                 e.printStackTrace();
          }
```

The new alarm severity profile replaces the default profile when required.

You can create several alarm severity profiles, each specifying a different response. One might designate fan failure as critical, another might designate high temperature as major. You then assign the appropriate profile to the object.

Clearing Alarms

Alarms are cleared automatically when each alarm relay is driven low. `OperationalState` will accordingly be shown to be `enabled`, `disabled` or `unknown`.

Software Monitoring

The following code examples help monitor software events. The series begins in CODE EXAMPLE 3-12 with establishing the printService to print system status reports, then gathers the list of software services and their associated daemons.

CODE EXAMPLE 3-12 Software Monitor Test (Part 1)

```
private void printService(ObjectName objName) {
    try {
        String name = (String)connectorClient.getAttribute(objName,"Name");
       String status = (String)connectorClient.getAttribute(objName,"Status");
        Integer polling_interval =\
           (Integer)connectorClient.getAttribute(objName,"PollingInterval");
        System.out.println("Name: " + name);
        System.out.println("Status: " + status);
        System.out.println("Polling Interval: " + polling_interval);
        }catch(Exception e) {
        e.printStackTrace();
        }
    }

         private void printDaemon(ObjectName objName) {
```

CODE EXAMPLE 3-12 Software Monitor Test (Part 1)

```
     try {
         String name = (String)connectorClient.getAttribute(objName,"Name");
         String status = (String)connectorClient.getAttribute(objName,"State");
         Integer retry = (Integer)connectorClient.getAttribute(objName,\
            "CurrentRetryCount");
         Integer maxretry = (Integer)connectorClient.getAttribute(objName,\
            "MaxRetryCount");

         System.out.println("name: " + name);
         System.out.println("state: " + status);
         System.out.println("retry: " + retry);
         System.out.println("maxretry: " + maxretry);
         }catch(Exception e) {e.printStackTrace();}
     }

     // This method traverses through the hierarchy of software monitor and
     // prints out all the software services and the daemons.
     private void test() {
         Object[] allObjs = null;
         Set swServiceList = null;
         ObjectName myObjName = null;
         try {
     // Get the list of all software services */
         swServiceList =(Set)connectorClient.invoke\
            (MohNames.MOH_SOFTWARE_MONITOR,"getSoftwareServiceList",null,null);
         allObjs = swServiceList.toArray();
```

CODE EXAMPLE 3-12 builds on previous examples to establish the status of **connectorClient**, and examine the hierarchy of the **swServiceList** in order to find existing services and running daemons.

The following segment of code collects the attributes of each software service so that the service can be registered to receive event notification.

CODE EXAMPLE 3-13 Traversing the Software Service List (Part 2)

```
      //Traverse through the software service list and print out the attributes
     //of each software service
     for (int i = 0; i < swServiceList.size();i++) {
         myObjName = (ObjectName)allObjs[i];
         System.out.println("service : "+ ((ObjectName)allObjs[i]).toString());
         printService(myObjName);
         // Register the software service to receive the event notifications
         connectorClient.addNotificationListener(myObjName,this,null,null);
```

CODE EXAMPLE 3-13 traverses **swServiceList** and adds **NotificationListener** to the **connectorClient**.

The final code segment gets the list of daemons that support the service, and prints out the daemon attributes for event notification.

CODE EXAMPLE 3-14 Getting the List of Service Daemons (Part 3)

```
          // For each service, get the list of daemons that support the service
          ObjectName[] daemonList =\
             (ObjectName[])connectorClient.getAttribute(myObjName,"DaemonList");
          if (daemonList != null && daemonList.length > 0){
              System.out.print("Daemon List:  ");
              Integer retry = null;
              Integer maxretry = null;
              // For each daemon, print out all attributes of the daemon.
              for (int k= 0;k < daemonList.length;k++) {
                  printDaemon(daemonList[k]);
                  // register the daemon to receive the event notifications
                  connectorClient.addNotificationListener\
                     (daemonList[k],this,null,null);
              }
          }
      }
      }catch(Exception e) {throw new UncheckedException(e);}
      }
```

CODE EXAMPLE 3-14 establishes a **DaemonList** for each service and prints out the attributes of each daemon. Finally, the code registers these daemons to receive notice of events with **addNotificationListener**.

For further information, look at `/opt/SUNWnetract/mgmt2.0/docs/api` which details all the MOH interfaces and classes that are provided for the Netra CT system software.

CHAPTER 4

Netra CT Element Management Agent API

This chapter contains the application programming interfaces (API) of the Netra CT element management agent software and includes the following sections:

Interface Overview

Netra CT management agent uses the Java Dynamic Management™ Kit (JDMK) framework as a Java API which provides the management capability for the Netra CT system.

JDMK supports JMX, which is a standard set of APIs for network/client management. JDMK provides an extended API along with different communication protocol adapters such as Remote Method Invocation (RMI), HTTP, HTML, and Simple Network Management Protocol (SNMP).

These protocol adapters are used to communicate with instances of JDMK agents; Netra CT management agent supports SNMP and RMI communication protocols.

You can find an introduction to the JDMK, tutorials, code samples, and APIs on the Java Developers website: `http://developer.java.sun.com`.

Summary of JDMK

JDMK's API and development tools can help you develop distributed management applications. The JDMK allows resources of one host to be monitored from another host.

A resource can be any entity, physical or virtual, that you wish to monitor through your network. Physical resources include network elements, and virtual resources include applications operating on a host. A resource can be seen through its management interface, where its attributes, operations, and notifications are accessible by a management agent.

In order for a management agent to monitor a resource, the resource must be developed as a managed bean (MBean), which is Java object that represents the resource's management interface. If the resource itself is a Java application, it can be its own MBean. Otherwise, an MBean is a Java representation of a device.

In the JDMK model, a Java Dynamic Management agent follows the client-server model, in which an agent responds to the management requests from any number of client applications that wish to access its resources. The central component of an agent is the MBean server, which is a registry for MBean instances and provides the framework that allows agent services to interact with MBeans.

The JDMK provides protocol connector interfaces that allow remote applications to access agent applications and their resources. Remote method invocation (RMI) and HTTP are two such JDMK supported protocols that enable a Java client application running on one system to access the resources and methods of another Java server application running on a different system.

FIGURE 4-1 displays the location of the RMI/HTTP protocols between an agent application and a remote manager application.

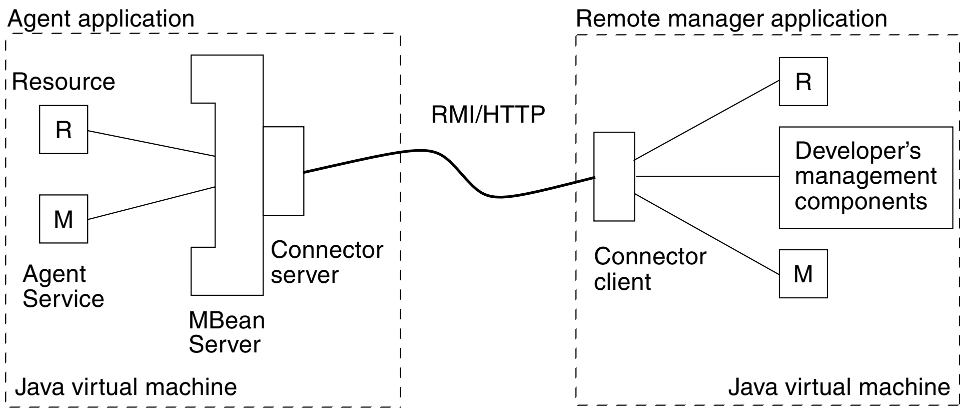


FIGURE 4-1 Key Components of the Java Dynamic Management Kit

In FIGURE 4-1, a resource and an agent service are registered as MBeans with the agent application's MBean server. The application agent also contains a connector server for the RMI/HTTP protocols. The remote manager application is a Java application running on a distant host system. The manager contains the RMI/HTTP connector client and proxy MBeans representing the resource and service. When the RMI/HTTP connector client establishes the connection with the agent's RMI/HTTP connector server, the other components of the application can issue management requests to the agent.

Typically, you would first determine the management interface of your resource, that is, the information needed to manage it. This information is expressed as attributes and operations. An attribute is a value of any type that a manager can get or set remotely. An operation is a method with any signature and any return type that the manager can invoke remotely.

As specified by the Java Management extensions for instrumentation, all attributes and operations are explicitly listed in an MBean interface. This Java interface defines the full management interface of an MBean. The interface must have the same name as the class that implements it, followed by the MBean suffix. Since the interface and its implementation are usually in different files, two files make up a standard MBean. For example, the management interface of the class SimpleStandard (in the file `SimpleStandard.java`) is defined in the interface SimpleStandardMBean (in the file `SimpleStandardMBean.java`).

For a complete discussion of JDMK components and protocols, refer to the Java Dynamic Management Kit documentation set found on the Solaris documentation website, `http://docs.sun.com`. For additional information of JDMK and the RMI/HTTP protocol, refer to the documentation, tutorials, code samples, and APIs found on the Java Developers website: `http://developer.java.sun.com`.

Viewing the Netra CT Management Agent API Online

The entire Netra CT RMI API specification can be viewed online as cross-referenced HTML pages. By default, these HTML pages are installed in the following directory:

```
/opt/SUNWnetract/mgmt2.0/docs/api/com/sun/ctmgx/moh
```

You can view an index of all of these pages by opening the following link in an web browser:

```
file:///opt/SUNWnetract/mgmt2.0/docs/api/index.html
```

You can view additional Java API specification on the `java.sun.com` webpage at:

```
http://java.sun.com/apis.html
```

How the API Sections are Organized

The following sections in this chapter list the classes of the Netra CT RMI application programming interface (API).

Each class, interface, inner class, and inner interface has its own separate section. Each of these sections have three subsections consisting of a class/interface description, summary tables, and detailed member descriptions of the following:

- Class inheritance diagram
- Direct subclasses
- All known subinterfaces
- All known implementing classes
- Class/interface declaration
- Class/interface description
- Inner class summary
- Field summary
- Constructor summary
- Method summary
- Field detail
- Constructor detail
- Method detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Netra CT Management Agent Interfaces and Classes

TABLE 4-1 lists the management agent interfaces and TABLE 4-2 lists the management agent classes included in the Netra CT RMI API. In these tables, the term *expose* refers to the encapsulation of the object's variables inside a nucleus. This encapsulation allows for exposing (allowing access to) or hiding (denying access to) an object's access methods, which provides for greater modularity.

Detailed descriptions of the interfaces begin with "Netra CT Management Agent Interface Descriptions" on page 56 and the detailed descriptions of the classes begin with "Netra CT Management Agent Class Descriptions" on page 136.

TABLE 4-1 Netra CT Management Agent Interfaces

Interface	Description	Refer to:
`AlarmCardPluginMBean`	Describes the management interface of the AlarmCardPluginMBean	"AlarmCardPluginMBean" on page 56.
`AlarmSeverityProfileMBean`	Describes the management interface of the AlarmSeverityProfileMBean.	"AlarmSeverityProfileMBean" on page 57.
`CgtpServiceMBean`	Describes the management interface of the CgtpServiceMBean.	"CgtpServiceMBean" on page 58.
`ContainmentTreeMBean`	Describes the management interface of the ContainmentTreeMBean.	"ContainmentTreeMBean" on page 60.
`CpiSlotMBean`	Describes the management interface of the CPCI Slot objects.	"CpciSlotMBean" on page 65.
`CpuCardEquipmentMBean`	Describes the management interface for the CPUCardEquipmentMBean.	"CpuCardEquipmentMBean" on page 67.
`CpuPluginMBean`	Describes the management interface of the CPU board objects as perceived from the alarm card MOH.	"CpuPluginMBean" on page 70.
`DaemonMBean`	Describes the interface for the DaemonMBean.	"DaemonMBean" on page 71.
`EFDMBean`	Describes the management interface of the EFDMBean.	"EFDMBean" on page 72.
`EquipmentHolderMBean`	Describes the management interface of the EquipmentHolderMBean.	"EquipmentHolderMBean" on page 73.
`EquipmentMBean`	Describes the interface of the EquipmentMBean.	"EquipmentMBean" on page 74.
`EtherIfStatsMBean`	Describes the management interface of the Ethernet I/O Statistics Monitoring service.	"EtherIfStatsMBean" on page 78.
`FullLogMBean`	Describes the interface of the FullLogMBean	"FullLogMBean" on page 80.
`IpServiceMBean`	Describes the interface of the Unix File System (UFS) service.	"IpServiceMBean" on page 86,
`LOLMBean`	Describes the management interface of the Latest Occurrence Log MBean	"LOLMBean" on page 88.
`NEMBean`	Describes the management interface of the NEMBean.	"NEMBean" on page 95.
`NetworkInterfaceMBean`	Describes the management interface for NetworkInterfaceMBean.	"NetworkInterfaceMBean" on page 98.

TABLE 4-1 Netra CT Management Agent Interfaces *(Continued)*

Interface	Description	Refer to:
`NfsServiceMBean`	Describes the management interface of the Network File System (NFS) Monitor service.	"NfsServiceMBean" on page 101.
`NumericSensorMBean`	Describes the interface for NumericSensorMBean.	"NumericSensorMBean" on page 104.
`PlugInUnitMBean`	Describes the management interface of the Plug-inUnitMBean.	"PlugInUnitMBean" on page 110.
`RnfsServiceMBean`	Describes the interface of the Reliable Network File System (RNFS) Monitor service.	"RnfsServiceMBean" on page 114.
`SensorMBean`	Describes the interface for the SensorMBean.	"SensorMBean" on page 116.
`SlotMBean`	Describes the management interface of the SlotMBean.	"SlotMBean" on page 118.
`SoftwareMonitorMBean`	Describes the interface of the SoftwareMonitorMBean.	"SoftwareMonitorMBean" on page 120.
`SoftwareServiceMBean`	Describes the interface of SoftwareServiceMBean.	"SoftwareServiceMBean" on page 124.
`TcpServiceMBean`	Describes the interface of Transmission Control Protocol (TCP) service.	"TcpServiceMBean" on page 127.
`TerminationPointMBean`	Describes the management interface of the TerminationPoint MBean.	"TerminationPointMBean" on page 130.
`UdpServiceMBean`	Describes the interface of User Datagram Protocol (UDP) service.	"UdpServiceMBean" on page 133.
`UfsServiceMBean`	Describes the interface of Unix File System (UFS) service.	"UfsServiceMBean" on page 134.

TABLE 4-2 Netra CT Management Agent Classes

Class	Description	Refer to:
`AdministrativeState`	Defines the administrative state of the device.	"AdministrativeState" on page 136.
`AlarmNotification`	The Alarm Notification class represents an alarm notification emitted by an MBean.	"AlarmNotification" on page 138.
`AlarmNotificationFilter`	This filter allows you to filter AlarmNotification notifications by selecting the types and severities of interest.	"AlarmNotificationFilter" on page 142.

TABLE 4-2 Netra CT Management Agent Classes *(Continued)*

Class	Description	Refer to:
`AlarmSeverity`	Defines the alarm severity objects for use with Alarm Notification.	"AlarmSeverity" on page 145.
`AlarmType`	This class is an enumeration of predefined Alarm types, user need to use one of the predefined types to construct an AlarmNotification object.	"AlarmType" on page 148.
`AttributeChangeNotification`	Provides definitions of the attribute change notifications sent by MBeans.	"AttributeChangeNotificatio n" on page 150.
`AttributeChangeNotificationFilter`	The filtering is performed on the name of the observed attribute.	"AttributeChangeNotificatio nFilter" on page 153.
`AuthClient`	This class defines the client utility routines, particularly for authentication.	"AuthClient" on page 154
`AvailabilityStatus`	Defines the availability status of the plug-in unit object.	"AvailabilityStatus" on page 156.
`EquipmentHolderType`	Describes the management interface of the EquipmentHolderType.	"EquipmentHolderType" on page 159.
`LogFullAction`	Describes the action to perform when the log is full.	"LogFullAction" on page 161
`MohNames`	Defines the public constants or static variables for MOH user to communicate to the MBean server.	"FullLogMBean" on page 80.
`ObjectCreationNotification`	Defines the creation notifications sent by MBeans.	"ObjectCreationNotification" on page 165.
`ObjectDeletionNotification`	Defines the deletion notifications sent by MBeans.	"ObjectDeletionNotification" on page 168.
`OperationalState`	Defines the operation states of a device (equipment or plug-in).	"OperationalState" on page 170.
`SlotStatus`	Defines the status of the slot object.	"SlotStatus" on page 172.
`StateChangeNotification`	Defines the state change notifications sent by MBeans.	"StateChangeNotification" on page 174.
`StateChangeNotificationFilter`	Describes the filtering performed on the name of the observed attribute.	"StateChangeNotificationFilt er" on page 177.

Netra CT Management Agent Interface Descriptions

This section contains descriptions of the following RMI interfaces listed in the order in which they appear in the source code:

AlarmCardPluginMBean

Declaration

`public interface` **`AlarmCardPluginMBean`** `extends PlugInUnitMBean`

Description

This class describes the management interface of the alarm card in a Netra CT system.

All Superinterfaces

`PlugInUnitMBean`

Method Summary	
`void`	`softReset()` Resets the alarm card with a graceful shutdown.

Inherited Member Summary

Methods inherited from interface `com.sun.ctmgx.moh.PlugInUnitMBean`

`getAdministrativeState, getAlarmSeverityProfilePointer, getAvailabilityStatus, getModelNumber, getOperationalState, getPlugInUnitLabel, getPlugInUnitType, getProductName, getSerialNumber, getVendorName, getVersion, setAdministrativeState, setAlarmSeverityProfilePointer`

Methods

`softReset`

`public void` **`softReset`**`()`

Resets the alarm card with a graceful shutdown.

AlarmSeverityProfileMBean

Declaration

`public interface` **`AlarmSeverityProfileMBean`**

Description

This class describes the management interface of the Alarm Severity Profile MBean.

Member Summary

Methods

`AlarmSeverity`	`getAlarmSeverity(AlarmType alarmType)` Gets the alarm severity assignment for a specific type of alarm notification.
`java.util.Map`	`getAlarmSeverityList()` Gets the alarm severity assignment list.
`void`	`setAlarmSeverity(AlarmType alarmType, AlarmSeverity severity)` Sets the alarm severity assignment for a specific type of alarm notification.

Methods

getAlarmSeverity(AlarmType)

```
public com.sun.ctmgx.moh.AlarmSeverity
   getAlarmSeverity(com.sun.ctmgx.moh.AlarmType alarmType)
```

Gets the alarm severity assignment for a specific type of alarm notification. This attribute identifies one alarm/severity pair.

Returns:

One of the following perceived severity values:

```
com.sun.ctmgx.moh.AlarmSeverity.CLEARED
com.sun.ctmgx.moh.AlarmSeverity.INDETERMINATE
com.sun.ctmgx.moh.AlarmSeverity.CRITICAL
com.sun.ctmgx.moh.AlarmSeverity.MAJOR
com.sun.ctmgx.moh.AlarmSeverity.MINOR
com.sun.ctmgx.moh.AlarmSeverity.WARNING
```

getAlarmSeverityList()

```
public java.util.Map getAlarmSeverityList()
```

Gets the alarm severity assignment list.

This attribute identifies one or more alarm/severity pairs.

Returns:

The alarm severity assignment list.

`setAlarmSeverity(AlarmType, AlarmSeverity)`

`public void` **`setAlarmSeverity`**`(com.sun.ctmgx.moh.AlarmType` *alarmType*`, com.sun.ctmgx.moh.AlarmSeverity` *severity*`)`

Sets the alarm severity assignment for a specific type of alarm notification.

This attribute identifies one alarm/severity pair.

Parameters:

alarmType – The alarm notification type.

severity – One of the perceived severity values defined as:

```
com.sun.ctmgx.moh.AlarmSeverity.CLEARED
com.sun.ctmgx.moh.AlarmSeverity.INDETERMINATE
com.sun.ctmgx.moh.AlarmSeverity.CRITICAL
com.sun.ctmgx.moh.AlarmSeverity.MAJOR
com.sun.ctmgx.moh.AlarmSeverity.MINOR
com.sun.ctmgx.moh.AlarmSeverity.WARNING
```

CgtpServiceMBean

Declaration

`public interface` **`CgtpServiceMBean`** `extends SoftwareServiceMBean`

Description

This class describes the interface of class CgtpService.

Member Summary

Methods

`java.lang.Float`	`getCurrentFilterThreshold()` Gets the current filtered failure threshold.
`java.lang.Float`	`getFilterMaxThreshold()` Gets the max filtered failure threshold.

Member Summary ***(Continued)***	
`java.lang.String[]`	`getIfList()` Get the list of Ethernet physical interfaces associated with CGTP service.
`OperationalState`	`getIfState(java.lang.String ifState)` Get the operational state of a CGTP physical interface.
`void`	`setFilterMaxThreshold(java.lang.Float threshold)` Sets the max filtered failure threshold.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

`getDaemonList(), getName(), getNumExcessiveIntervals(), getPollingInterval(), getStatus(), setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling()`

Methods

getCurrentFilterThreshold()

`public java.lang.Float` **`getCurrentFilterThreshold()`**

Gets the current filtered failure threshold.

Returns:

The threshold in percentage of the filtered failure packets over the total received packets.

getFilterMaxThreshold()

`public java.lang.Float` **getFilterMaxThreshold()**

Gets the max filtered failure threshold.

Returns:

The threshold in percentage of the filtered failure packets over the total received packets.

getIfList()

`public java.lang.String[]` **`getIfList()`**

Get the list of Ethernet physical interfaces associated with CGTP service.

Returns:

A string array of interface names.

`getIfState`

`public OperationalState` **`getIfState`**`(java.lang.String ifState)`

Get the operational state of a CGTP physical interface.

Parameters:

The CGTP physical interface.

Returns:

OperationalState.ENABLED, or OperationalState.DISABLED, or OperationalState.UNKNOWN.

`setFilterMaxThreshold(Float)`

`public void` **setFilterMaxThreshold**`(java.lang.Float` *threshold*`)`

Sets the max filtered failure threshold.

Parameters:

threshold — threshold to set.

ContainmentTreeMBean

Declaration

`public interface` **`ContainmentTreeMBean`**

Description

This class describes the management interface of the Containment Tree MBean.

Member Summary

Fields

`static java.lang.String`	`CT_FILTER_EQUIPMENT` Gets the Equipment object(s) only during topology discovery.
`static java.lang.String`	`CT_FILTER_HOLDER` Gets the Holder object(s) only during topology discovery.
`static java.lang.String`	`CT_FILTER_NE` Gets the NE object(s) only during topology discovery.
`static java.lang.String`	`CT_FILTER_NETWORK_INTERFACE` Gets the Network Interface object(s) only during topology
`static java.lang.String`	`CT_FILTER_PLUGIN_UNIT` Gets the Plug-in object(s) only during topology discovery.

Member Summary *(Continued)*	
`static java.lang.String`	`CT_FILTER_TERMINATION_POINT` Gets the Termination Point object(s) only during topology discovery.
Methods	
`java.util.Set`	`getAncestors(javax.management.ObjectName node)` Gets the ancestors of the specified MBean in the containment hierarchy.
`java.util.Set`	`getAncestors(javax.management.ObjectName node, java.lang.String filter)` Gets the ancestors of the specified MBean in the containment hierarchy that match the given filter type.
`java.util.Set`	`getChildren(javax.management.ObjectName node)` Gets the children of the specified MBean in the containment hierarchy.
`java.util.Set`	`getChildren(javax.management.ObjectName node, java.lang.String filter)` Gets the children of the specified MBean in the containment hierarchy that match the given type.
`java.util.Set`	`getDescendants(javax.management.ObjectName node)` Gets the descendants of the specified MBean in the containment hierarchy.
`java.util.Set`	`getDescendants(javax.management.ObjectName node, java.lang.String filter)` Gets the descendants of the specified MBean in the containment hierarchy that match the given type.
`javax.management.ObjectName`	`getParent(javax.management.ObjectName node)` Gets the parent of the specified MBean in the containment hierarchy.
`javax.management.ObjectName`	`getParent(javax.management.ObjectName node, java.lang.String filter)` Gets the parent of the specified MBean in the containment hierarchy, if it matches the given type.
`javax.management.ObjectName`	`getRoot()` Gets the object name of the root MBean in the containment hierarchy.

Fields

CT_FILTER_EQUIPMENT

`public static final java.lang.String` **`CT_FILTER_EQUIPMENT`**

This attribute gets the Equipment object(s) only during topology discovery.

`CT_FILTER_HOLDER`

`public static final java.lang.String` **`CT_FILTER_HOLDER`**

This attribute gets the Holder object(s) only during topology discovery.

`CT_FILTER_NE`

`public static final java.lang.String` **`CT_FILTER_NE`**

This attribute gets the NE object(s) only during topology discovery.

`CT_FILTER_NETWORK_INTERFACE`

`public static final java.lang.String` **`CT_FILTER_NETWORK_INTERFACE`**

This attribute is used to get the Network Interface object(s) only during topology discovery.

`CT_FILTER_PLUGIN_UNIT`

`public static final java.lang.String` **`CT_FILTER_PLUGIN_UNIT`**

This attribute gets the Plugin object(s) only during topology discovery.

`CT_FILTER_TERMINATION_POINT`

`public static final java.lang.String` **`CT_FILTER_TERMINATION_POINT`**

This attribute gets the Termination Point object(s) only during topology discovery.

Methods

`getAncestors(ObjectName)`

`public java.util.Set` **`getAncestors`**`(javax.management.ObjectName` *node*`)`

Gets the ancestors of the specified MBean in the containment hierarchy.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

Returns:

A set containing the `ObjectName` objects for the ancestors MBeans. If the specified MBean has no ancestors an empty list is returned.

`getAncestors(ObjectName, String)`

`public java.util.Set` **`getAncestors`**`(javax.management.ObjectName` *node*`, java.lang.String` *filter*`)`

Gets the ancestors of the specified MBean in the containment hierarchy that match the given filter type.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

filter – The filter constant used for filtering the retrieved MBeans.

Returns:

A set containing the `ObjectName` objects for the ancestors MBeans. If the specified MBean has no ancestors or if the ancestors do not match the given type an empty list is returned.

`getChildren(ObjectName)`

`public java.util.Set` **`getChildren`**`(javax.management.ObjectName` *node*`)`

Gets the children of the specified MBean in the containment hierarchy.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

Returns:

A set containing the `ObjectName` objects for the children MBeans. If the specified MBean has no children, an empty list is returned.

`getChildren(ObjectName, String)`

`public java.util.Set` **`getChildren`**`(javax.management.ObjectName` *node*`, java.lang.String` *filter*`)`

Gets the children of the specified MBean in the containment hierarchy that match the given filter type.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

filter – The filter constant used for filtering the retrieved MBeans.

Returns:

A set containing the `ObjectName` objects for the children MBeans. If the specified MBean has no children, or if the children do not match the given type, an empty list is returned.

`getDescendants(ObjectName)`

`public java.util.Set getDescendants(javax.management.ObjectName` *node*`)`

Gets the descendants of the specified MBean in the containment hierarchy.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

Returns:

A set containing the `ObjectName` objects for the descendants MBeans. If the specified MBean has no descendants an empty list is returned.

`getDescendants(ObjectName, String)`

`public java.util.Set` **`getDescendants`**`(javax.management.ObjectName` *node*`, java.lang.String` *filter*`)`

Gets the descendants of the specified MBean in the containment hierarchy that match the given type.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

filter – The filter constant used for filtering the retrieved MBeans.

Returns:

A set containing the `ObjectName` objects for the descendants MBeans. If the specified MBean has no descendants or if the descendants do not match the given type an empty list is returned.

`getParent(ObjectName)`

`public javax.management.ObjectName` **`getParent`**`(javax.management.ObjectName` *node*`)`

Gets the parent of the specified MBean in the containment hierarchy.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

Returns:

The `ObjectName` object for the parent MBean. If the specified MBean has no parent a `null` value is returned.

`getParent(ObjectName, String)`

`public javax.management.ObjectName` **`getParent`**`(javax.management.ObjectName` *node*`, java.lang.String` *filter*`)`

Gets the parent of the specified MBean in the containment hierarchy, if it matches the given type.

Parameters:

node – The `ObjectName` of the MBean where the search should start from.

filter – The filter constant used for filtering the retrieved MBeans.

Returns:

The `ObjectName` object for the parent MBean. If the specified MBean has no parent or if its parent does not match the given type, a `null` value is returned.

`getRoot()`

`public javax.management.ObjectName` **`getRoot`**`()`

Gets the object name of the root MBean in the containment hierarchy.

Returns:

The `ObjectName` object for the root MBean. This is the object name of the instance of NEMBean.

CpciSlotMBean

Declaration

`public interface` **`CpciSlotMBean`** extends `SlotMBean`

Description

This class describes the management interface of the CPCI Slot objects.

All Superinterfaces

`EquipmentHolderMBean`, `SlotMbean`

Member Summary	
Fields	
`static Boolean`	`OFF` Powered off state
`static Boolean`	`ON` Powered on state
Methods	
`Boolean`	`getPowerState()` Gets slot power state PowerState can be either ON or OFF.
`void`	`hardReset()` Executes a hard reset the board in this slot.
`void`	`setPowerState(Boolean powerState)` Sets the slot's PowerState.

Inherited Member Summary
Methods inherited from interface `com.sun.ctmgx.moh.SlotMBean`
`getAcceptablePlugInUnitTypes, getSlotStatus, getSlotType, getSoftwareLoad, setAcceptablePlugInUnitTypes, setSoftwareLoad`
Methods inherited from `interface` `com.sun.ctmgx.moh.EquipmentHolderMBean`
`getEquipmentHolderAddress, getEquipmentHolderLabel, getEquipmentHolderType`

Fields

`OFF`

`public static final Boolean` **`OFF`**

Powered off state

`ON`

`public static final Boolean` **`ON`**

Powered on state

Methods

`getPowerState()`

`public Boolean` **`getPowerState()`**

Gets the slot's power state. The PowerState can be either `ON` or `OFF`. This state shows whether the board in this CPCI slot is powered on or not.

Returns:

The PowerState value which can be one of:

```
com.sun.ctmgx.moh.CpciSlotMBean.ON
com.sun.ctmgx.moh.CpciSlotMBean.OFF
```

`setPowerState(boolean powerState)`

`public void` **`setPowerState`**`(Boolean` *powerState*`)`

Sets the slot's PowerState.

Parameters:

powerState is one of:

```
com.sun.ctmgx.moh.CpciSlotMBean.ON
com.sun.ctmgx.moh.CpciSlotMBean.OFF
```

`hardReset()`

`public void` **`hardReset()`**

`hardReset``(com.sun.ctmgx.moh.CpciSlotMBean)`

Hard resets the board in this slot.

CpuCardEquipmentMBean

Declaration

`public interface` **`CpuCardEquipmentMBean`** extends `EquipmentMBean.`

All Superinterfaces

`EquipmentMBean`

Description

This class describes the management interface for CPU Card Equipment MBean.

CpuCardEquipmentMBean Memory Monitor Feature:

The purpose of this feature is to monitor memory usage on this CPU. There are three memory usage thresholds which are settable by user: Minor memory usage threshold, Major memory usage threshold and Critical threshold.

Minor and Major memory usage thresholds are represented in percentage of memory usage over total physical memory. Critical threshold is based on number of memory pages paged out per second.

The defaults are: Minor threshold 94%, Major threshold 97%, and Critical threshold, 1 page paged out per second.

An alarm will occur when physical memory utilization exceeds one of the specified thresholds. If the memory utilization/paging decreases below a specified threshold for a specified time interval, the alarm will clear. `void`

Member Summary

Methods

`java.lang.Integer`	`getClearAlarmTimeOut()` Gets time required for a memory alarm to clear.
`java.lang.Integer`	`getThreshCriticalMemoryUsed()` Provides the critical threshold of memory utilization in number of page-outs per second, the default value is 1 page per second.

Member Summary	
`java.lang.Integer`	`getThreshMajorMemoryUsed()` Provides the major threshold of memory utilization in percentage value, the default value is 97%.
`java.lang.Integer`	`getThreshMinorMemoryUsed()` Provides the minor threshold of memory utilization in percentage value, the default value is 94%.
`void`	`setClearAlarmTimeOut(java.lang.Integer timeOutPeriod)` Sets time required for a memory alarm to clear.
`void`	`setThreshCriticalMemoryUsed(java.lang.Integer pagesPerSec)` Sets the critical threshold of memory utilization in terms of number of page-outs per second.
`void`	`setThreshMajorMemoryUsed(java.lang.Integer pctUsed)` Gets the major threshold of memory utilization in percentage value.
`void`	`setThreshMinorMemoryUsed(java.lang.Integer pctUsed)` Sets the minor threshold of memory utilization in percentage value.
`void`	`softReset()` This resets the CPU

Inherited Member Summary

Methods inherited from interface `EquipmentMBean`

`getAdministrativeState(), getAlarmSeverityProfilePointer(), getEquipmentType(), getLocationName(), getModelNumber, getOperationalState(), getProductName, getSerialNumber, getUserLabel(), getVendorName(), getVersion(), setAdministrativeState(AdministrativeState), setAlarmSeverityProfilePointer(ObjectName), setLocationName(String), setUserLabel(String)`

Methods

`getClearAlarmTimeOut()`

`public java.lang.Integer` **`getClearAlarmTimeOut`**()

Gets time required for a memory alarm to clear.

Returns:

Time in seconds.

`getThreshCriticalMemoryUsed()`

`public java.lang.Integer` **`getThreshCriticalMemoryUsed`**()

Provides the critical threshold of memory utilization in number of page-outs per second. The default value is 1 page per second.

Returns:

The threshold in number of page-outs per second.

`getThreshMajorMemoryUsed()`

`public java.lang.Integer` **`getThreshMajorMemoryUsed`**()

Provides the major threshold of memory utilization in percentage value. The default value is 97%.

Returns:

Major threshold in percentage.

`getThreshMinorMemoryUsed()`

`public java.lang.Integer` **`getThreshMinorMemoryUsed`**()

Sets the minor threshold of memory utilization in percentage value.

Parameters:

pctUsed – value in percentage to set

`setClearAlarmTimeOut(Integer)`

`public void` **`setClearAlarmTimeOut`**(`java.lang.Integer` *timeOutPeriod*)

Set time required for a memory alarm to clear.

Parameters:

timeOutPeriod – value in seconds.

`setThreshCriticalMemoryUsed(Integer)`

`public void` **`setThreshCriticalMemoryUsed`**(`java.lang.Integer` pagesPerSec)

Sets the critical threshold of memory utilization in terms of number of page-outs per second.

Parameters:

pagesPerSec – value in page-outs per second to set.

`setThreshMajorMemoryUsed(Integer)`

`public void` **`setThreshMajorMemoryUsed`**(`java.lang.Integer` *pctUsed*)

`setThreshMinorMemoryUsed(Integer)`

`public void` **`setThreshMinorMemoryUsed`**(`java.lang.Integer` pctUsed)

This lets the user get the major threshold of memory utilization in percentage value.

Parameters:

pctUsed – value in percentage to set.

`softReset()`

`public void` **`softReset()`**

This resets the CPU

CpuPluginMBean

Declaration

`public interface` **`CpuPluginMBean`** `extends PlugInUnitMBean`

All Superinterfaces

`PlugInUnitMBean`

Description

This class describes the management interface of the CPU board objects as perceived from the alarm card MOH.

Member Summary

Methods

`void`	`softReset()` This lets user to reset this CPU board with a graceful shutdown of the operating system running on it.

Inherited Member Summary

Methods inherited from interface `PlugInUnitMBean`

`getAdministrativeState, getAlarmSeverityProfilePointer, getAvailabilityStatus, getModelNumber, getOperationalState, getPlugInUnitLabel, getPlugInUnitType, getProductName, getSerialNumber, getVendorName, getVersion, setAdministrativeState, setAlarmSeverityProfilePointer`

Methods

`softReset()`

`public void softReset()`

Resets this CPU board with a graceful shutdown of the operating system.

DaemonMBean

Declaration

`public interface DaemonMBean`

This class describes the interface for `DaemonMBean`.

Member Summary	
Methods	
`java.lang.Integer`	`getCurrentRetryCount()` Gets the current number of retries to recover the daemon.
`java.lang.Integer`	`getMaxRetryCount()` Gets the maximum number of retries to recover the daemon
`java.lang.String`	`getName()` Gets the name of the daemon.
`java.lang.String`	`getState()` Gets the state of the daemon.
`void`	`setMaxRetryCount(java.lang.Integer maxRetries)` Set the maximum recovery retry count for a daemon.

Methods

`getCurrentRetryCount()`

`public java.lang.Integer getCurrentRetryCount()`

Gets the current number of retries to recover the daemon.

Returns:

Number of times the system has tried to recover the daemon.

`getMaxRetryCount()`

`public java.lang.Integer getMaxRetryCount()`

Gets the maximum number of retries to recover the daemon.

Returns:

Maximum number of times the system will try to recover the daemon.

`getName()`

`public java.lang.String` **`getName()`**

Gets the name of the daemon.

Returns:

String indicating name of the daemon. For example, on Solaris, Platform Management Service daemon name is `picld`, NFS service is `nfsd`.

`getState()`

`public java.lang.String` **`getState()`**

Gets the state of the daemon.

This attribute identifies the state of the daemon.

Returns:

The daemon states values defined as string`("running")` if the daemon is running, or string`("configured")` if the daemon is not running and is installed, or string`("unconfigured")`, if the daemon is not running and is not installed.

`setMaxRetryCount(Integer)`

`public void` **`setMaxRetryCount`**(`java.lang.Integer` *maxRetries*)

Set the maximum recovery retry count for a daemon.

Indicates how many times the system will try to recover the daemon.

Parameters:

maxRetries – The number of retries.

EFDMBean

Declaration

`public interface` **`EFDMBean`**

Description

This class describes the management interface of the EFD MBean.

EquipmentHolderMBean

Declaration

`public interface` **`EquipmentHolderMBean`**

All Known Subinterfaces

`CpciSlotMBean,SlotMBean`

Description

This class describes the management interface of the EquipmentHolderMBean

Member Summary

Methods	
`java.lang.String`	`getEquipmentHolderAddress()` Gets the equipment holder address.
`java.lang.String`	`getEquipmentHolderLabel()` Gets the equipment holder label.
`EquipmentHolderType`	`getEquipmentHolderType()` Gets the equipment holder type.

Methods

`getEquipmentHolderAddress()`

`public java.lang.String` **`getEquipmentHolderAddress`**`()`

Gets the equipment holder address.

This attribute identifies the physical location of the resource represented by the Equipment Holder instance.

Returns:

The equipment holder address.

`getEquipmentHolderLabel()`

`public java.lang.String` **`getEquipmentHolderLabel`**`()`

Gets the equipment holder label.

This attribute identifies the external label string for this equipment holder.

Returns:

The equipment holder’s external label string, if none, a null string(“”) is returned.

`getEquipmentHolderType()`

`public com.sun.ctmgx.moh.EquipmentHolderType` **`getEquipmentHolderType`**`()`

Gets the equipment holder type.

Indicates whether the Equipment Holder instance is being used to represent a rack, shelf, drawer, or slot.

Returns:

The equipment holder type defined as:

```
com.sun.ctmgx.moh.EquipmentHolderType.RACK
com.sun.ctmgx.moh.EquipmentHolderType.SHELF
com.sun.ctmgx.moh.EquipmentHolderType.DRAWER
com.sun.ctmgx.moh.EquipmentHolderType.SLOT
```

EquipmentMBean

Declaration

`public interface` **`EquipmentMBean`**

All Known Subinterfaces:

`CpuCardEquipmentMBean, NumericSensorMBean, SensorMBean`

Description

This class describes the interface of the Equipment MBean.

Member Summary

Methods

`AdministrativeState`	`getAdministrativeState()` Gets the administrative state.
`javax.management.ObjectName`	`getAlarmSeverityProfilePointer()` Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.
`java.lang.String`	`getEquipmentType()` Gets the equipment type.
`java.lang.String`	`getLocationName()` Get the location name.

Member Summary *(Continued)*

`java.lang.String`	`getModelNumber()` Get the part number.
`OperationalState`	`getOperationalState()` Get the operational state.
`java.lang.String`	getProductName() Get the product name.
`java.lang.String`	getSerialNumber() Get the serial number.
`java.lang.String`	`getUserLabel()` Get the user label.
`java.lang.String`	`getVendorName()` Get the vendor name.
`java.lang.String`	`getVersion()` Gets the version.
`void`	`setAdministrativeState(AdministrativeState admin_state)` Sets the administrative state.
`void`	`setAlarmSeverityProfilePointer(javax.management.ObjectName asp)` Sets the ObjectName of the AlarmSeverityProfile associated with this MBean.
`void`	`setLocationName(java.lang.String location_name)` Sets the location name.
`void`	`setUserLabel(java.lang.String user_label)` Sets the user label.

Methods

`getAdministrativeState()`

`public com.sun.ctmgx.moh.AdministrativeState` **`getAdministrativeState`**`()`

Gets the administrative state.

This attribute is used to activate (unlocked) and deactivate (locked) the function performed by the Equipment.

Returns:

The administrative state value defined as either:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`getAlarmSeverityProfilePointer()`

public javax.management.ObjectName **getAlarmSeverityProfilePointer**()

Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments is used.

Returns:

The ObjectName of the AlarmSeverityProfile MBean.

`getEquipmentType()`

public java.lang.String **getEquipmentType**()

Get the equipment type `String`.

This attribute identifies the type of the equipment.

Returns:

The equipment type of the EquipmentMBean instance.

`getLocationName()`

public java.lang.String **getLocationName**()

Get the location name.

This attribute identifies the specific or general location of the Equipment.

Returns:

The specific or general location of the Equipment.

`getOperationalState()`

public com.sun.ctmgx.moh.**OperationalState** **getOperationalState**()

Gets the operational state.

This attribute identifies whether or not the Equipment is capable of performing its normal functions, that is, enabled or disabled.

Returns:

One of the following operational state values:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

getUserLabel()

`public java.lang.String` **`getUserLabel`**`()`

Gets the user label.

This attribute gets a user-friendly name to the associated Equipment.

Returns:

The user label of the Equipment.

getVendorName()

`public java.lang.String` **`getVendorName`**`()`

Gets the vendor name.

This attribute identifies the vendor of the Equipment.

Returns:

The vendor of the Equipment.

getVersion()

`public java.lang.String` **`getVersion`**`()`

Gets version.

This attribute identifies the version of the Equipment.

Returns:

The version of the Equipment.

setAdministrativeState(AdministrativeState)

`public void` **`setAdministrativeState`**`(com.sun.ctmgx.moh.Administrat iveState` *admin_state*`)`

Sets the administrative state.

This attribute is used to activate (unlocked) and deactivate (locked) the function performed by the Equipment.

Parameters:

admin_state – The administrative state value defined as either:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

setAlarmSeverityProfilePointer(ObjectName)

`public void` **`setAlarmSeverityProfilePointer`**`(javax.management.Obje ctName` *asp*`)`

Sets the ObjectName of the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments shall be used.

Parameters:

asp – The ObjectName of the AlarmSeverityProfile MBean.

`setLocationName(String)`

`public void` **`setLocationName`**`(java.lang.String` *location_name*`)`

Sets the location name.

This attribute identifies the specific or general location of the Equipment.

Parameters:

location_name – The location name.

`setUserLabel(String)`

`public void` **`setUserLabel`**`(java.lang.String` *user_label*`)`

Sets the user label.

This attribute is used to assign a user friendly name to the associated Equipment.

Parameters:

user_label – The user label of the Equipment.

EtherIfStatsMBean

Declaration

`public interface` **`EtherIfStatsMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the management interface of Ethernet I/O Statistics Monitoring service.

Member Summary	
Methods	
`java.lang.Float`	`getMaxThreshhold()` Gets the threshold in percentage of receive-transmit errors.
`java.lang.String[]`	`getRxExcessiveThreshholdIfList()` Gets the list of Ethernet interfaces exceeding the receive error threshold.
`java.lang.String[]`	`getTxExcessiveThreshholdIfList()` Gets the list of Ethernet interfaces exceeding the transmit error threshold.
`void`	`setMaxThreshhold(java.lang.Float threshhold)` Sets the threshold in percentage of receive-transmit errors.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

`getDaemonList(), getName(), getNumExcessiveIntervals(), getPollingInterval(), getStatus(), setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling(), stopPolling()`

Methods

getMaxThreshhold()

`public java.lang.Float` **`getMaxThreshhold`**`()`

Gets the threshold in percentage of receive-transmit errors.

This attribute identifies the receive and transmit error threshhold.

Returns:

Percentage value in Float format.

getRxExcessiveThreshholdIfList()

`public java.lang.String[]` **`getRxExcessiveThreshholdIfList`**`()`

Gets the list of Ethernet interfaces exceeding the receive error threshold

Returns:

The list of Ethernet interfaces OR null if there is none.

`getTxExcessiveThreshholdIfList()`

`public java.lang.String[]` **`getTxExcessiveThreshholdIfList`**`()`

Gets the list of Ethernet interfaces exceeding the transmit error threshold.

Returns:

The list of Ethernet interfaces OR null if there is none.

`setMaxThreshhold(Float)`

`public void` **`setMaxThreshhold`**`(java.lang.Float` *threshhold*`)`

Sets the threshold in percentage of receive-transmit errors.

This attribute identifies the receive and transmit error threshhold.

Parameters:

threshold — The percentage to set.

FullLogMBean

Declaration

`public interface` **`FullLogMBean`**

Description

This class describes the management interface of FullLog MBean. This MBean is used to group multiple instances of the Notification classes to form a log of Notifications.

Instances of this MBean are created automatically by the NE upon initialization. Creation and deletion of instances of this MBean from the management system are not allowed.

Upon initialization of the MOH agent, there will be just one instance of FullLog which is initialized with a default filter, all notifications from MOH agent will be logged. The Object name for this instance is defined by `MohNames.OBJECT_NAME_FULL_LOG`.

Notifications emitted by this MBean:

- `com.sun.ctmgx.moh.ObjectCreationNotification.OBJECT_CREATION`

  This notification is used to report the creation of an instance of this MBean.

- `com.sun.ctmgx.moh.ObjectDeletionNotification.OBJECT_DELETION`

  This notification is used to report the deletion of an instance of this MBean.

- `com.sun.ctmgx.moh.AttributeChangeNotification.ATTRIBUTE_CHANGE`

  This notification is used to report changes to the LogFullAction attribute of this MBean. The notification identifies the attribute that changed, its old value, and its new value.

- `com.sun.ctmgx.moh.StateChangeNotification.STATE_CHANGE`

  This notification is used to report changes to the OperationalState attribute and AdministrativeState attribute of this MBean. The notification identifies the state attribute that changed, its old value, and its new value.

Member Summary

Methods	
`void`	deleteAllRecords() Deletes all records logged in the Log till now.
`void`	deleteRecords(int noOfRecords, int startIndex) Deletes records from the Log starting from a specified starting index.
`AdministrativeState`	getAdministrativeState() Get the administrative state.
`java.util.Vector`	getAllRecords() Get all the records logged.
`int`	`getCapacity()` Get the capacity or the maximum value of the number of records which can be logged.
`javax.management.NotificationFilter`	`getFilter()` Get the filter which is used to filter the Notifications to be logged.
`LogFullAction`	`getLogFullAction()` Get the log full action.
`java.util.Vector`	`getLogRecordTypes()` Get a vector of Notification types being logged.
`OperationalState`	`getOperationalState()` Get the operational state.
`java.util.Vector`	`getRecords(int noOfRecords, int startIndex)` Get the records from a starting index.
`int`	`getSize()` Get the total number of records logged.

Member Summary *(Continued)*

`void`	`setAdministrativeState(AdministrativeState admin_state)` Set the administrative state.
`void`	`setFilter(javax.management.NotificationFilter logFilter)` Set the filter, which will be used to filter the Notifications to be logged.
`void`	`setLogFullAction(LogFullAction logFullAction)` Set the log full action.

Methods

`deleteAllRecords()`

`public void` **`deleteAllRecords`**`()`

Deletes all records logged in the Log up to this point.

`deleteRecords(int, int)`

`public void` **`deleteRecords`**`(int` *noOfRecords*`, int` *startIndex*`) throws jav a.lang.ArrayIndexOutOfBoundsException`

Deletes records from the Log starting from a specified starting index.

Parameters:

noOfRecords – The total number of records to be deleted.

startIndex – The index in the log starting from which the records will be deleted.

Throws:

Throws – `ArrayIndexOutOfBoundsException` if *startIndex* points beyond 0..size-1.

`getAdministrativeState()`

`public com.sun.ctmgx.moh.AdministrativeState` **`getAdministrativeSt ate`**`()`

Gets the administrative state.

This attribute is used to activate (unlocked) and deactivate (locked) the function performed by the Log.

Returns:

The administrative state value defined as one of the following:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`getAllRecords()`

`public java.util.Vector` **`getAllRecords`**`()`

Gets all the records logged

Returns:

A vector containing all the records logged up to this point.

`getCapacity()`

`public int` **`getCapacity`**`()`

Gets the capacity or the maximum value of the number of records that can be logged.

Returns:

The capacity of the log.

`getFilter()`

`public javax.management.NotificationFilter` **`getFilter`**`()`

Gets the filter which is used to filter the Notifications to be logged.

Returns:

The `NotificationFilter` object.

`getLogFullAction()`

`public com.sun.ctmgx.moh.LogFullAction` **`getLogFullAction`**`()`

Gets the log full action.

This attribute is used to identify the action the NE takes when the log space is full. The valid values for this attribute are `wrap around` and `halt`. The default value is `wrap around`.

Returns:

The log full action value defined as one of the following:
`com.sun.ctmgx.moh.LogFullAction.WRAP`
`com.sun.ctmgx.moh.LogFullAction.HALT`

`getLogRecordTypes()`

`public java.util.Vector` **`getLogRecordTypes`**`()`

Gets a vector of Notification types being logged.

This attribute identifies the types of log records grouped by this log-managed entity.

Returns:

A vector object containing the types as String defined as `com.sun.ctmgx.moh.AlarmNotification.FAN_FAILURE`, `com.sun.ctmgx.moh.AlarmNotification.FUSE_FAILURE`, `com.sun.ctmgx.moh.AlarmNotification.HIGH_TEMPERATURE`, and so on.

`getOperationalState()`

`public com.sun.ctmgx.moh.OperationalState` **`getOperationalState`**`()`

Gets the operational state.

This attribute identifies whether or not the Log is capable of performing its normal functions, that is, enabled or disabled.

Returns:

The operational state value defined as one of the following:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

`getRecords(int, int)`

`public java.util.Vector` **`getRecords`**`(int` *noOfRecords,* `int` *startIndex*`)`
`throws java.lang.ArrayIndexOutOfBoundsException`

Gets the records from a starting index

Parameters:

noOfRecords – The total number of records to be retrieved.

startIndex – Index in the log starting from which the records will be retrieved.

Returns:

A vector containing *noOfRecords* number of records.

Throws:

Throws – `ArrayIndexOutOfBoundsException` if *startIndex* points beyond 0..size-1.

`getSize()`

`public int` **`getSize`**`()`

Gets the total number of records logged.

Returns:

The size of the log.

`setAdministrativeState(AdministrativeState)`

`public void` **`setAdministrativeState`**`(com.sun.ctmgx.moh.AdministrativeState` *admin_state*`)`

Sets the administrative state.

This attribute activates (unlocks) and deactivates (locks) the function performed by the Log.

Parameters:

admin_state – The administrative state value defined as one of the following:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`setFilter(NotificationFilter)`

`public void` **`setFilter`**`(javax.management.NotificationFilter` *logFilter*`)`

Sets the filter, for the Notifications to be logged.

Parameters:

logFilter – An object of type NotificationFilter. This can be object of one of the following:

```
com.sun.ctmgx.moh.AlarmNotificationFilter
com.sun.ctmgx.moh.AttributeChangeNotificationFilter
com.sun.ctmgx.moh.StateChangeNotificationFilter
javax.management.NotificationFilterSupport
```

`setLogFullAction(LogFullAction)`

`public void` **`setLogFullAction`**`(com.sun.ctmgx.moh.LogFullAction` *logFullAction*`)`

Sets the log full action.

This attribute is used to identify the action the NE takes when the log space is full. The valid values for this attribute are `wrap around` and `halt`. The default value is `wrap`.

Parameters:

logFullAction – the log full action value defined as one of the following:

```
com.sun.ctmgx.moh.LogFullAction.WRAP
com.sun.ctmgx.moh.LogFullAction.HALT
```

IpServiceMBean

Declaration

`public interface` **`IpServiceMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the interface of Unix File System (UFS) service.

Member Summary

Methods

`java.lang.Float`	`getCurrentInAddrErrorThreshold()` Gets the current threshold of input datagrams discarded due to invalid IP address in their IP header's destination field.
`java.lang.Float`	`getCurrentInHdrErrorThreshold()` Gets the current threshold of input datagrams discarded due to errors in their IP headers.
`java.lang.Float`	`getMaxInAddrErrorThreshold()` Gets the maximum threshold of input datagrams discarded due to an invalid IP address in IP header destination field.
`java.lang.Float`	`getMaxInHdrErrorThreshold()` Gets the maximum threshold of input datagrams discarded due to an error in the IP header.
`void`	`setMaxInAddrErrorThreshold(java.lang.Float newThreshold)` Sets the maximum threshold of input datagrams discarded due to an invalid IP address in IP header.
`void`	`setMaxInHdrErrorThreshold(java.lang.Float newThreshold)` Sets the maximum threshold of input datagrams discarded due to errors in their IP headers.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

```
getDaemonList(), getName(), getNumExcessIntervals(),
getPollingInterval(),getStatus(), setNumExcessiveIntervals(Integer),
setPollingInterval(Integer), startPolling(), stopPolling()
```

Methods

`getCurrentInAddrErrorThreshold()`

`public java.lang.Float` **getCurrentInAddrErrorThreshold**`()`

Gets the current threshold of input datagrams discarded due to an invalid IP address in their IP header's destination field.

Returns:

The threshold in percentage of number of errors over the total receiving datagrams.

`getCurrentInHdrErrorThreshold()`

`public java.lang.Float` **getCurrentInHdrErrorThreshold**`()`

Gets the current threshold of input datagrams discarded due to errors in their IP headers.

Returns:

The threshold in percentage of number of errors over the total receiving datagrams.

`getMaxInAddrErrorThreshold()`

`public java.lang.Float` **getMaxInAddrErrorThreshold()**

Gets the maximum threshold of input datagrams discarded due to invalid IP address in IP header destination field.

Returns:

The maximum threshold in percentage of number of errors over the total receiving datagrams.

`getMaxInHdrErrorThreshold()`

`public java.lang.Float` **getMaxInHdrErrorThreshold()**

Gets the maximum threshold of input datagrams discarded due to an error in the IP header.

Returns:

The maximum threshold in percentage of number of errors over the total receiving datagrams.

`setMaxInAddrErrorThreshold(Float)`

`public void` **setMaxInAddrErrorThreshold(java.lang.Float** *newThreshold*`)`

Sets the maximum threshold of input datagrams discarded due to invalid IP address in IP header.

Parameters:

newThreshold – maximum threshold to set.

`setMaxInHdrErrorThreshold(Float)`

`public void` **setMaxInHdrErrorThreshold(java.lang.Float** *newThreshold*`)`

Sets the maximum threshold of input datagrams discarded due to errors in their IP headers.

Parameters:

newThreshold – maximum threshold to set.

LOLMBean

Declaration

`public interface` **`LOLMBean`**

Description

This class describes the management interface for Latest Occurrence Log MBean.

This MBean groups multiple instances of the Notification classes to form a latest occurrence log (LOL) of Notifications.

If no other log Notification/record contained in LOL has values of the attributes identified by the 'key attribute list' attribute equal to the attribute values of the log record to be added, the log record is added as a new entry to LOL. If a log record contained in the LOL has values of the attributes identified by the 'key attribute list' attribute equal to the attribute values of the log record to be added, the older existing log record contained in the LOL is replaced by the new log record.

Instances of this MBean are created automatically by the NE upon initialization.

Creation and deletion of instances of this MBean from the management system are not allowed.

Upon initialization of the MOH agent there are two instances of this class. One instance is initialized with an `AlarmNotificationFilter` with all severities enabled and 'Type' and 'PerceivedSeverity' being set as the Key attributes. The AdministrativeState is also activated. The object name for this instance is defined by `NetraCtDefs.OBJECT_NAME_LOL_ALARM`.

And the other instance is just initialized with a default filter for `netract.moh` The object name for this instance is defined by `NetraCtDefs.OBJECT_NAME_LOL_ALL.`

Notifications emitted by this MBean:

- `com.sun.ctmgx.moh.ObjectCreationNotification.OBJECT_CREATION`

  This notification is used to report the creation of an instance of this MBean.

- `com.sun.ctmgx.moh.ObjectDeletionNotification.OBJECT_DELETION`

  This notification is used to report the deletion of an instance of this MBean.

- `com.sun.ctmgx.moh.AttributeChangeNotification.ATTRIBUTE_CHANGE`

  This notification is used to report changes to the LogFullAction attribute of this MBean. The notification identifies the attribute that changed, its old value, and its new value.

- `com.sun.ctmgx.moh.StateChangeNotification.STATE_CHANGE`

  This notification is used to report changes to the OperationalState attribute and AdministrativeState attribute of this MBean. The notification identifies the state attribute that changed, its old value, and its new value.

Method Summary	
`void`	`addAttribute(java.lang.String` *attribute*`)` Adds the attribute into the key attribute list.
`void`	`deleteAllRecords()` Deletes all records logged in the log till now.
`void`	`deleteRecords(int` *noOfRecords*`, int` *startIndex*`)` Deletes records from the Log starting from a specified starting index.
`AdministrativeState`	`getAdministrativeState()` Gets the administrative state.
`java.util.Vector`	`getAllRecords()` Gets all logged records.
`int`	`getCapacity()` Gets the capacity or the maximum value of the number of records that can be logged.
`javax.management.NotificationFilter`	`getFilter()` Gets the filter used to filter the Notifications to be logged.
`java.util.Set`	`getKeyAttributeList()` Gets the key attribute list as an object of Set.
`LogFullAction`	`getLogFullAction()` Gets the log full action.
`java.util.Vector`	`getLogRecordTypes()` Gets a set of Notification types being logged.

Method Summary *(Continued)*

`OperationalState`	`getOperationalState()` Gets the operational state.
`java.util.Vector`	`getRecords(int` *noOfRecords*`, int` *startIndex*`)` Gets the records from a starting index.
`int`	`getSize()` Gets the total number of records logged.
`void`	`removeAttribute(java.lang.String` *attribute*`)` Removes the attribute.
`void`	`setAdministrativeState(AdministrativeState` *admin_state*`)` Sets the administrative state.
`void`	`setCapacity(int capacity)` Sets the capacity.
`void`	`setFilter(javax.management.NotificationFilter logFilter)` Sets the filter, which will be used to filter the Notifications to be logged.
`void`	`setKeyAttributeList(java.util.Set` *list*`)` Sets the key attribute list.
`void`	`setLogFullAction(LogFullAction logFullAction)` Sets the log full action.

Methods

addAttribute(String)

`public void` **`addAttribute`**`(java.lang.String` *attribute*`)`

Adds the attribute into the key attribute list.

Parameters:

attribute – Name (string) of the attribute to be added to the key attribute list

deleteAllRecords()

`public void` **`deleteAllRecords`**`()`

Deletes all records logged in the Log till now.

deleteRecords(int, int)

`public void` **`deleteRecords`**`(int` *noOfRecords*`, int` *startIndex*`)`
`throws ArrayIndexOutOfBoundsException`

Deletes records from the Log starting from a specified starting index.

Parameters:

noOfRecords – The total number of records to be deleted

startIndex – Index in the log starting from which the records will be deleted

Throws:

Throws – `ArrayIndexOutOfBoundsException` if *startIndex* points beyond 0..size-1.

`getAdministrativeState()`

`public com.sun.ctmgx.moh.AdministrativeState` **`getAdministrativeState`**`()`

Gets the administrative state.

This attribute is used to activate (unlock) and deactivate (lock) the function performed by the Log.

Returns:

The administrative state value defined as one of the following:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`getAllRecords()`

`public java.util.Vector` **`getAllRecords`**`()`

Gets all the records logged.

Returns:

A vector containing all the records logged till now.

`getCapacity()`

`public int` **`getCapacity`**`()`

Gets the capacity or the maximum value of the number of records which can be logged.

Returns:

The capacity of the log.

`getFilter()`

`public javax.management.NotificationFilter` **`getFilter`**`()`

Gets the filter which is used to filter the Notifications to be logged.

Returns:

The `NotificationFilter` object.

`getKeyAttributeList()`

`public java.util.Set` **`getKeyAttributeList`**`()`

Gets the key attribute list as an object of Set.

Returns:

An object of `java.util.Set` containing all the key attribute names as String.

`getLogFullAction()`

`public com.sun.ctmgx.moh.LogFullAction` **`getLogFullAction`**`()`

Gets the log full action.

This attribute is used to identify the action the NE takes when the log space is full. The valid values for this attribute are `wrap around` and `halt`. The default value is `wrap around`.

Returns:

The log full action value defined as one of the following:

```
com.sun.ctmgx.moh.LogFullAction.WRAP
com.sun.ctmgx.moh.LogFullAction.HALT
```

`getLogRecordTypes()`

`public java.util.Vector` **`getLogRecordTypes`**`()`

Gets a set of Notification types being logged.

This attribute identifies the types of log records grouped by this log-managed entity.

Returns:

A set object containing the types as String defined as `com.sun.ctmgx.moh.AlarmNotification.FAN_FAILURE`, `com.sun.ctmgx.moh.AlarmNotification.FUSE_FAILURE`, `com.sun.ctmgx.moh.AlarmNotification.HIGH_TEMPERATURE`, and so on.

`getOperationalState()`

`public com.sun.ctmgx.moh.OperationalState` **`getOperationalState`**`()`

Gets the operational state.

This attribute identifies whether or not the Log is capable of performing its normal functions, that is, enabled or disabled.

Returns:

The operational state value defined as one of the following:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

`getRecords(int, int)`

`public java.util.Vector` **`getRecords`**`(int` *noOfRecords*`, int` *startIndex*`) throws ArrayIndexOutOfBoundsException`

Gets the records from a starting index.

Parameters:

noOfRecords –The total number of records to be retrieved.

startIndex – The index in the log starting from which the records will be retrieved.

Returns:

A vector containing *noOfRecord* number of records.

Throws:

Throws – `ArrayIndexOutOfBoundsException` if *startIndex* points beyond 0..size-1.

`getSize()`

`public int` **`getSize`**`()`

Gets the total number of records logged.

Returns:

The size of the log.

`removeAttribute(String)`

`public void` **`removeAttribute`**`(java.lang.String` *attribute*`)`

Remove the attribute.

Parameters:

attribute – A String which is the name of the attribute to be deleted from the key attribute list.

`setAdministrativeState(AdministrativeState)`

`public void` **`setAdministrativeState`**`(com.sun.ctmgx.moh.AdministrativeState` *admin_state*`)`

Sets the administrative state.

This attribute is used to activate (unlock) and deactivate (lock) the function performed by the Log.

Parameters:

admin_state – The administrative state value defined as one of the following:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`setCapacity(int)`

`public void` **`setCapacity`**`(int` *capacity*`)`

Sets the capacity.

Parameters:

capacity – Maximum size for the log.

`setFilter(NotificationFilter)`

`public void` **`setFilter`**`(javax.management.NotificationFilter` *logFilter*`)`

Sets the filter, which will be used to filter the Notifications to be logged.

Parameters:

logFilter – An object of type NotificationFilter. This can be object of one of the following:

```
com.sun.ctmgx.moh.AlarmNotificationFilter
com.sun.ctmgx.moh.AttributeChangeNotificationFilter
com.sun.ctmgx.moh.StateChangeNotificationFilter
javax.management.NotificationFilterSupport
```

`setKeyAttributeList(Set)`

`public void` **`setKeyAttributeList`**`(java.util.Set` *list*`)`

Sets the key attribute list.

This attribute indicates the list of attribute names to be used as keys to uniquely identify the entries in a latest occurrence log.

Parameters:

list – An object of java.util.Set containing the attribute names

`setLogFullAction(LogFullAction)`

`public void` **`setLogFullAction`**`(com.sun.ctmgx.moh.LogFullAction` *logFullAction*`)`

Sets the log full action.

This attribute is used to identify the action the NE takes when the log space is full. The valid values for this attribute are `wrap` and `halt`. The default value is `wrap`.

Parameters:

logFullAction – The log full action value defined as one of the following:

```
com.sun.ctmgx.moh.LogFullAction.WRAP
com.sun.ctmgx.moh.LogFullAction.HALT
```

NEMBean

Declaration

`public interface` **`NEMBean`**

Description

This class describes the management interface of the NE MBean.

Member Summary

Methods

`javax.management.ObjectName`	`getAlarmSeverityProfilePointer()` Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.

`java.lang.String`	`getDescription()`
	`Get Description of the NE.`

`long`	`getExternalTime()` This attribute provides the time-of-day system time.
`java.lang.String`	`getLocationName()` Gets the location name.
`OperationalState`	`getOperationalState()` Gets the operational state.
`java.lang.Boolean`	`getSecurityState()` Get the security state of the NE.
`java.lang.String`	`getVendorName()` Gets the vendor name.

Member Summary *(Continued)*

`java.lang.String`	`getVersion()` Gets the version.
`void`	`setAlarmSeverityProfilePointer(ObjectName asp)` Sets the ObjectName of the AlarmSeverityProfile associated with this MBean.
`void`	`setLocationName(java.lang.String location_name)` Sets the location name.

Methods

getAlarmSeverityProfilePointer()

`public javax.management.ObjectName` **`getAlarmSeverityProfilePointer`**`()`

Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments are used.

Returns:

The ObjectName of the AlarmSeverityProfile MBean.

getDescription()

`public java.lang.String` **getDescription()**

Gets description of the NE.

Returns:

Description of this network element.

getExternalTime()

`public long` **`getExternalTime`**`()`

Gets the time-of-day system time.

Returns:

The difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

getLocationName()

`public java.lang.String` **`getLocationName`**`()`

Gets the location name of the NE.

This attribute identifies the specific or general location of the NE.

Returns:

The specific or general location of the NE.

`getOperationalState()`

`public com.sun.ctmgx.moh.OperationalState` **`getOperationalState`**`()`

Gets the operational state.

This attribute identifies whether or not the NE MBean is capable of performing its normal functions, that is, enabled or disabled.

Returns:

One of the following operational state values:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

`getSecurityState`

`public java.lang.Boolean` **`getSecurityState()`**

Get the security state of the NE.

This attribute identifies the security state of the NE. It is used to determine whether authentication checks need to be performed on the client connection. A boolean value of **true** indicates that authentication checking will be performed on each client connection. (See "Netra CT Agent Security" on page 32 for more information.)

Returns:

The SecurityState of the NE.

`getVendorName()`

`public java.lang.String` **`getVendorName`**`()`

Gets the vendor name.

This attribute identifies the vendor of the NE.

Returns:

The vendor of the NE.

`getVersion()`

`public java.lang.String` **`getVersion`**`()`

Gets the version.

This attribute identifies the version of the NE. A value of `null` should be used in cases where version information is not available or applicable to the NE being represented.

Returns:

The version of the NE.

`setAlarmSeverityProfilePointer(ObjectName)`

`public void` **`setAlarmSeverityProfilePointer`**`(ObjectName` *asp*`)`

Sets the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments shall be used.

Parameters:

asp – The ObjectName of the AlarmSeverityProfile MBean.

`setLocationName(String)`

`public void` **`setLocationName`**`(java.lang.String` *location_name*`)`

Sets the location name.

Parameters:

location_name – The location name to set

NetworkInterfaceMBean

Declaration

`public interface` **`NetworkInterfaceMBean`**

Description

This class describes the management interface for Network Interface MBean.

This interface needs to be implemented to represent a network interface object in the system.

Member Summary

Methods

`javax.management.ObjectName`	`getAlarmSeverityProfilePointer()` Gets the AlarmSeverityProfile associated with this MBean.
`java.lang.Integer`	`getMtu()` Gets Mtu (Maximum Transmit Unit) of the interface
`java.lang.String`	`getName()` Gets the name of the interface
`OperationalState`	`getOperationalState()` Gets the operational state.
`java.lang.String`	`getPermanentAddress()` Gets address of the interface
`javax.management.ObjectName`	`getTerminationPoint` Gets the physical TerminationPoint used by this network interface
`java.lang.String`	`getType()` Gets interface type
`java.lang.String`	`getVendorName()` Gets the vendor name.
`java.lang.String`	`getVersion()` Gets the version.
`void`	`setAlarmSeverityProfilePointer(ObjectName asp)` Sets the AlarmSeverityProfile associated with this MBean.

Methods

`getAlarmSeverityProfilePointer()`

```
public ObjectName getAlarmSeverityProfilePointer()
```

Gets the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments shall be used.

Returns:

The ObjectName of the AlarmSeverityProfile MBean.

`getMtu()`

`public java.lang.Integer` **`getMtu`**`()`

Gets Mtu (Maximum Transmit Unit) of the interface.

Returns:

The maximum size of a packet this interface can send/receive.

`getName()`

`public java.lang.String` **`getName`**`()`

Gets the name of the interface.

Returns:

The name of the network interface.

`getOperationalState()`

`public com.sun.ctmgx.moh.OperationalState` **`getOperationalState`**`()`

Gets the operational state.

Returns:

One of the following operational state values:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

`getPermanentAddress()`

`public java.lang.String` **`getPermanentAddress`**`()`

Gets the address of the interface.

Returns:

The device specific physical address.

`getTerminationPoint()`

`public javax.management.ObjectName` **`getTerminationPoint()`**

Gets the physical TerminationPoint used by this network interface.

Returns:

The ObjectName of the TerminationPoint.

`getType()`

`public java.lang.String` **`getType`**`()`

Gets interface type.

Returns:

Type of the network interface.

`getVendorName()`

`public java.lang.String` **`getVendorName`**`()`

Gets the vendor name.

Returns:

Vendor name.

`getVersion()`

`public java.lang.String` **`getVersion`**`()`

Gets the version.

Returns:

Version.

`setAlarmSeverityProfilePointer(ObjectName)`

`public void`
`setAlarmSeverityProfilePointer``(javax.management.ObjectName` **`asp`**`)`

Sets the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments shall be used.

Parameters:

asp – The ObjectName of the AlarmSeverityProfile MBean.

NfsServiceMBean

Declaration

`public interface` **`NfsServiceMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the management interface of the Network File System (NFS) Monitor service.

Member Summary

Methods

`java.lang.Float`	`getClientMaxThreshHold()` Gets client maximum threshhold.
`java.lang.Float`	`getClientThreshHold()` Gets client current threshhold.
`java.lang.String[]`	`getMountFailureList()` Gets the list of failed NFS mount points
`java.lang.Float`	`getServerMaxThreshHold()` Gets server maximum threshhold.
`java.lang.Float`	`getServerThreshHold()` Gets server current threshhold.
`void`	`setClientMaxThreshHold(java.lang.Float threshHold)` Sets the client max threshhold.
`void`	`setServerMaxThreshHold(java.lang.Float threshHold)` Sets the server maximum threshhold.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

`getDaemonList(), getName(), getNumExcessiveIntervals(), getPollingInterval(), getStatus(), setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling(), stopPolling()`

Methods

getClientMaxThreshHold()

`public java.lang.Float` **`getClientMaxThreshHold`**`()`

Gets client maximum threshhold. Threshhold is the percentage of errors/total transactions.

Returns:

Threshold in percentage.

`getClientThreshHold()`

`public java.lang.Float` **`getClientThreshHold`**`()`

Gets client current threshhold.

Threshhold is the percentage of errors/total transactions

Returns:

Threshold in percentage.

`getMountFailureList()`

`public java.lang.String[]` **`getMountFailureList`**`()`

Gets the list of failed NFS mount points

Returns:

List of failed file system mount points.

`getServerMaxThreshHold()`

`public java.lang.Float` **`getServerMaxThreshHold`**`()`

Gets server maximum threshhold.

Threshhold is the percentage of errors/total transactions.

Returns:

Threshold in percentage.

`getServerThreshHold()`

`public java.lang.Float` **`getServerThreshHold`**`()`

Gets server current threshhold.

Threshhold is the percentage of errors/total transactions.

Returns:

Threshold in percentage.

`setClientMaxThreshHold(Float)`

`public void` **`setClientMaxThreshHold`**`(java.lang.Float` *threshHold*`)`

Sets client maximum threshhold.

Threshhold is the percentage of errors/total transactions.

Parameters:

`threshHold`–threshold to set.

The default value is 0.

`setServerMaxThreshHold(Float)`

`public void` **`setServerMaxThreshHold`**`(java.lang.Float` *threshHold*`)`

Sets server maximum threshhold.

Threshhold is the percentage of errors/total transactions.

Parameters:

threshHold–threshold to set.

The default value is 0.

NumericSensorMBean

Declaration

`public interface` **`NumericSensorMBean`** extends `SensorMBean`

All Superinterfaces

`EquipmentMBean, SensorMBean`

Description

This class describes the interface for NumericSensorMBean.

This Mbean is used to model the thermistor (temperature sensor) on the CPU card. There are three temperature regions to monitor: noncritical, critical, and fatal. Each of the temperature regions has lower and upper boundaries or thresholds.

In general, an alarm is generated when the temperature falls below the lower threshold or exceeds the upper threshold.

Not all thresholds are supported in Netra CT. In Netra C 410 and CT 810, only the upper threshold of each region is supported.

The noncritical and critical thresholds are settable and under user control. The fatal threshold is the CPU junction temperature which is determined and set at board manufacturing time. This threshold is read-only by user.

An alarm occurs when the temperature exceeds the noncritical or critical threshold.

An alarm occurs and the system goes through automatic shutdown when the temperature exceeds the fatal threshold.

Member Summary

Fields	
static java.lang.Short	`CRITICAL_LOWER` Lower bound of the second (middle) level threshold.
static java.lang.Short	`CRITICAL_UPPER` Upper bound of the second (middle) level threshold.
static java.lang.Short	`FATAL_LOWER` Lower bound of the third (highest) level threshold.
static java.lang.Short	`FATAL_UPPER` Upper bound of the third (highest) level threshold.
static java.lang.Short	`NONCRITICAL_LOWER` Lower bound of the first (lowest) level threshold.
static java.lang.Short	`NONCRITICAL_UPPER` Upper bound of the first (lowest) level threshold.
Methods	
java.lang.String	`getBaseUnits()` Gets the base unit of this numeric sensor.
java.lang.Short[]	`getEnabledThresholds()` Gets enabled thresholds.
java.lang.Integer	`getLowerThresholdCritical()` Gets lower critical threshold of sensor-reading.
java.lang.Integer	`getLowerThresholdFatal()` Gets upper fatal threshold of sensor-reading.
java.lang.Integer	`getLowerThresholdNonCritical()` Gets lower non-critical threshold of sensor-reading.
java.lang.String	`getRateUnits()` If the RateUnits property is set to a value other than `none`, the units are further qualified as rate units.
java.lang.Short[]	`getSettableThresholds()` Gets settable thresholds.
java.lang.Short[]	`getSupportedThresholds()` Gets supported thresholds for this system.
java.lang.Integer	`getUnitModifier()`
java.lang.Integer	`getUpperThresholdCritical()` Gets upper critical threshold of sensor-reading.

Member Summary *(Continued)*

`java.lang.Integer`	`getUpperThresholdFatal()` Gets upper fatal threshold of sensor-reading.
`java.lang.Integer`	`getUpperThresholdNonCritical()` Gets the upper non-critical threshold of sensor-reading.
`void`	`setUpperThresholdCritical(java.lang.Integer threshVal)` Sets the upper critical threshold to a new value.
`void`	`setUpperThresholdNonCritical(java.lang.Integer threshVal)` Sets the upper non-critical threshold to a new value.

Inherited Member Summary

Fields inherited from interface `SensorMBean`

`SENSOR_TYPE_OTHER, SENSOR_TYPE_TEMPERATURE`

Methods inherited from interface `EquipmentMBean`

`getAdministrativeState(), getAlarmSeverityProfilePointer(), getEquipmentType(), getLocationName(), getModelNumber, getOperationalState(), getProductName, getSerialNumber, getUserLabel(), getVendorName(), getVersion(), setAdministrativeState(AdministrativeState), setAlarmSeverityProfilePointer(ObjectName), setLocationName(String), setUserLabel(String)`

Methods inherited from interface `SensorMBean`

`getCurrentState(), getPossibleStates(), getSensorType()`

Fields

CRITICAL_LOWER

```
public static final java.lang.Short CRITICAL_LOWER
```

Lower bound of the second (middle) level threshold.

CRITICAL_UPPER

```
public static final java.lang.Short CRITICAL_UPPER
```

Upper bound of the second (middle) level threshold.

FATAL_LOWER

```
public static final java.lang.Short FATAL_LOWER
```

Lower bound of the third (highest) level threshold.

FATAL_UPPER

`public static final java.lang.Short` **`FATAL_UPPER`**

Upper bound of the third (highest) level threshold.

NONCRITICAL_LOWER

`public static final java.lang.Short` **`NONCRITICAL_LOWER`**

Lower bound of the first (lowest) level threshold.

NONCRITICAL_UPPER

`public static final java.lang.Short` **`NONCRITICAL_UPPER`**

Upper bound of the first level threshold.

Methods

getBaseUnits()

`public java.lang.String` **`getBaseUnits`**`()`

Gets the base unit of this numeric sensor; e.g., a temperature sensor might read the temperature in degree C.

Returns:

`degree C` for temperature in Celsius.

getEnabledThresholds()

`public java.lang.Short[]` **`getEnabledThresholds`**`()`

Gets enabled thresholds. This API provides the same information as `getSupportedThresholds`.

Returns:

An array of enabled thresholds where each element(Short) in the array identifies a threshold type. An instance of this MBean, which is a temperature sensor, might return a Short array { NONCRITICAL_UPPER, CRITICAL_UPPER, FATAL_UPPER }.

getLowerThresholdCritical()

`public java.lang.Integer` **`getLowerThresholdCritical`**`()`

Gets lower critical threshold of sensor-reading.

Returns:

Lower threshold.

`getLowerThresholdFatal()`

`public java.lang.Integer` **`getLowerThresholdFatal`**`()`

Gets upper fatal threshold of sensor-reading.

Returns:

Upper threshold.

`getLowerThresholdNonCritical()`

`public java.lang.Integer` **`getLowerThresholdNonCritical`**`()`

Gets lower non-critical threshold of sensor-reading.

Returns:

Lower threshold.

`getRateUnits()`

`public java.lang.String` **`getRateUnits`**`()`

If the RateUnits property is set to a value other than `none`, the units are further qualified as rate units. For an instance that is a temperature sensor, the value is `none`.

Returns:

`none` for temperature sensor.

`getSettableThresholds()`

`public java.lang.Short[]` **`getSettableThresholds`**`()`

Gets settable thresholds.

Returns:

An array of settable thresholds where each element(Short) in the array identifies a threshold type. An instance of this MBean, which is a temperature sensor, might return a Short array {NONCRITICAL_UPPER, CRITICAL_UPPER}. All possible thresholds modelled by this NumericSensor are declared as public constants.

`getSupportedThresholds()`

`public java.lang.Short[]` **`getSupportedThresholds`**`()`

Gets supported thresholds for this system. This API provides the same information as getEnabledThresholds.

Returns:

An array of supported thresholds where each element(Short) in the array identifies a threshold type. An instance of this MBean, which is a temperature sensor, might return a Short array {NONCRITICAL_UPPER, CRITICAL_UPPER, FATAL_UPPER}.

All possible thresholds modelled by this NumericSensor are declared as public constants.

`getUnitModifier()`

`public java.lang.Integer` **`getUnitModifier`**`()`

`getUpperThresholdCritical()`

`public java.lang.Integer` **`getUpperThresholdCritical`**`()`

Gets upper critical threshold of sensor-reading.

Returns:

Upper threshold.

`getUpperThresholdFatal()`

`public java.lang.Integer` **`getUpperThresholdFatal`**`()`

Gets upper fatal threshold of sensor-reading.

Returns:

Upper threshold.

`getUpperThresholdNonCritical()`

`public java.lang.Integer` **`getUpperThresholdNonCritical`**`()`

Gets the upper noncritical threshold of sensor-reading.

Returns:

Upper threshold.

`setUpperThresholdCritical(Integer)`

`public void` **`setUpperThresholdCritical`**`(java.lang.Integer` *threshVal*`)`

Sets the upper critical threshold to a new value.

Parameters:

threshVal– Value of temperature

`setUpperThresholdNonCritical(Integer)`

`public void` **`setUpperThresholdNonCritical`**`(java.lang.Integer` *threshVal*`)`

Sets the upper noncritical threshold to a new value.

Parameters:

threshVal– Value of temperature.

PlugInUnitMBean

Declaration

`public interface` **`PlugInUnitMBean`**

All Known Subinterfaces:

`AlarmCardPluginMBean,CpuPluginMBean`

Description

This class describes the management interface of the Plug-in Unit MBean.

Member Summary

Methods

`AdministrativeState`	`getAdministrativeState()` Gets the administrative state.
`javax.management.ObjectName`	`getAlarmSeverityProfilePointer()` Gets the AlarmSeverityProfile associated with this MBean.
`AvailabilityStatus`	`getAvailabilityStatus()` Gets the availability status.

`java.lang.String`	`getModelNumber()`
	`Get the part number.`

`OperationalState`	`getOperationalState()` Gets the operational state.
`java.lang.String`	`getPlugInUnitLabel()` Gets the external label string, if there is any.
`java.lang.String`	`getPlugInUnitType()` Gets the plug-in unit type.

Member Summary *(Continued)*	

`java.lang.String`	`getProductName()`
	`Get the product name.`

`java.lang.String`	`getSerialNumber()`
	`Get the serial number.`

`java.lang.String`	`getVendorName()` Gets the vendor name.
`java.lang.String`	`getVersion()` Gets the version.
`void`	`setAdministrativeState(AdministrativeState admin_state)` Sets the administrative state.
`void`	`setAlarmSeverityProfilePointer(javax.management.ObjectName asp)` Sets the AlarmSeverityProfile associated with this MBean.

Methods

getAdministrativeState()

`public com.sun.ctmgx.moh.AdministrativeState` **`getAdministrativeState`**`()`

Gets the administrative state.

This function gets the current status(locked/unlocked) of this object.

Returns:

The administrative state value defined as either:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

getAlarmSeverityProfilePointer()

`public javax.management.ObjectName` **`getAlarmSeverityProfilePointer`**`()`

Gets the AlarmSeverityProfile associated with this MBean.

This function gets a pointer to an instance of the Alarm Severity Profile MBean containing the severity assignments for the alarms reported by this MBean. When the returned value is `null`, the default alarm severity profile is used.

Returns:

The ObjectName of the AlarmSeverityProfile MBean.

`getAvailabilityStatus()`

`public com.sun.ctmgx.moh.AvailabilityStatus` **`getAvailabilityStatus`**`()`

Gets the availability status.

This function is used to get the availability status of this object. Valid values for the availability status are Available, In Test, Failed, Power Off, Not Installed, Off Line, and Dependency. This last value indicates that the plug-in unit cannot operate because some other resource on which it depends is unavailable.

Returns:

One of the following availability status values:

```
com.sun.ctmgx.moh.AvailabilityStatus.AVAILABLE
com.sun.ctmgx.moh.AvailabilityStatus.INTEST
com.sun.ctmgx.moh.AvailabilityStatus.FAILED
com.sun.ctmgx.moh.AvailabilityStatus.POWEROFF
com.sun.ctmgx.moh.AvailabilityStatus.NOTINSTALLED
com.sun.ctmgx.moh.AvailabilityStatus.OFFLINE
com.sun.ctmgx.moh.AvailabilityStatus.DEPENDENCY
```

`getOperationalState()`

`public com.sun.ctmgx.moh.OperationalState` **`getOperationalState`**`()`

Gets the operational state.

This function gets the operation state of the object. The operational state indicates whether or not the plug-in unit is capable of performing its normal functions, i.e. enabled or disabled.

Returns:

One of the following operational state values:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

`getPlugInUnitLabel()`

`public java.lang.String` **`getPlugInUnitLabel`**`()`

Gets the external label string, if there is any.

Returns:

The external label string of the plug-in unit.If there is no label, a null string(“”) is returned.

`getPlugInUnitType()`

`public java.lang.String` **`getPlugInUnitType`**`()`

Gets the plug-in unit type.

Returns:

The plug-in unit type.

`getVendorName()`

`public java.lang.String` **`getVendorName`**`()`

Gets the vendor name.

Returns:

The vendor name of the plug-in unit.

`getVersion()`

`public java.lang.String` **`getVersion`**`()`

Gets the version.

Returns:

The version of the plug-in unit.

`setAdministrativeState(AdministrativeState)`

`public void` **`setAdministrativeState`**`(com.sun.ctmgx.moh.AdministrativeState` *admin_state*`)`

Sets the administrative state.

This function sets the administrative status of this object.

Parameters:

admin_state – The administrative state value defined as either `com.sun.ctmgx.moh.AdministrativeState.LOCKED` or `com.sun.ctmgx.moh.AdministrativeState.UNLOCKED`.

`setAlarmSeverityProfilePointer(ObjectName)`

`public void` **`setAlarmSeverityProfilePointer`**`(ObjectName` *asp*`)`

Sets the AlarmSeverityProfile associated with this MBean.

This function sets the pointer attribute of this Mbean to the instance of the Alarm Severity Profile. When the setting value is `null`, the default alarm severity profile is used.

Parameters:

asp – The ObjectName of the AlarmSeverityProfile object.

RnfsServiceMBean

Declaration

`public interface` **`RnfsServiceMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the interface of the Reliable Network File System (RNFS) Monitor service.

Member Summary

Methods

`java.lang.String[]`	`getExcessiveThresholdFileSystemList()` Gets the list of RNFS exceeding threshold.
`java.lang.Float`	`getMaxSyncThreshold()` Gets the maximum synchronization threshold.
`java.lang.Float`	`getSyncThreshold(java.lang.String rnfs)` Gets the current synchronization threshhold of a RNFS.
`void`	`setMaxSyncThreshold(java.lang.Float threshold)` Sets the maximum synchronization threshold of all RNFS.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

```
setDaemonList(),getName(),getNumExcessiveIntervals(),getPollingInterval(),
getStatus(),setNumExcessiveIntervals(Integer), setPollingInterval(Integer),
startPolling(), stopPolling()
```

Methods

`getExcessiveThresholdFileSystemList()`

`public java.lang.String[]` **getExcessiveThresholdFileSystemList()**

Gets the list of RNFS exceeding threshold.

Returns:

List of file system.

Null returned if there is no file systems exceeding threshold.

`getMaxSyncThreshold()`

`public java.lang.Float` **getMaxSyncThreshold()**

Gets maximum synchronization threshold.

Threshold is the percentage of RNFS disk space that needs to be synchronized.

Returns:

Threshold in percentage.

`getSyncThreshold(String)`

`public java.lang.Float` **getSyncThreshold(java.lang.String rnfs)**

Gets current synchronization threshhold of a RNFS.

Threshold is the percentage of RNFS disk space that needs to be synchronized.

Returns:

Threshold in percentage.

`setMaxSyncThreshold(Float)`

`public void` **setMaxSyncThreshold**`(java.lang.Float` *threshold*`)`

Sets maximum synchronization threshold of all RNFS.

Threshold is the percentage of RNFS disk space that needs to be synchronized.

Parameters:

threshold – threshold to set.

The default value is 0.

SensorMBean

Declaration

`public interface` **`SensorMBean`** extends `EquipmentMBean`

All Superinterfaces

`EquipmentMBean`

All Known Subinterfaces

`NumericSensorMBean`

Description

This class describes the interface for Sensor MBean. For Netra CT 410 and CT 810, only sensor type SENSOR_TYPE_TEMPERATURE is supported.

Member Summary	
Fields	
`static java.lang.Short`	`SENSOR_TYPE_OTHER` Other sensor typ.e
`static java.lang.Short`	`SENSOR_TYPE_TEMPERATURE` Temperature sensor.
Methods	
`java.lang.String`	`getCurrentState()` Gets the current sensor state.
`java.lang.String[]`	`getPossibleStates()` Gets the sensor possible states.
`java.lang.Short`	`getSensorType()` Gets the sensor type.

Inherited Member Summary

Methods inherited from interface `EquipmentMBean`

`getAdministrativeState(), getAlarmSeverityProfilePointer(), getEquipmentType(), getLocationName(), getModelNumber, getOperationalState(), getProductName, getSerialNumber, getUserLabel(), getVendorName(), getVersion(), setAdministrativeState(AdministrativeState), setAlarmSeverityProfilePointer(ObjectName), setLocationName(String), setUserLabel(String)`

Fields

SENSOR_TYPE_OTHER

`public static final java.lang.Short` **`SENSOR_TYPE_OTHER`**

Other sensor type.

SENSOR_TYPE_TEMPERATURE

`public static final java.lang.Short` **`SENSOR_TYPE_TEMPERATURE`**

Temperature sensor.

Methods

getCurrentState()

`public java.lang.String` **`getCurrentState`**`()`

Gets current sensor state.

Returns:

The current state of the sensed property. Currently not supported.

getPossibleStates()

`public java.lang.String[]` **`getPossibleStates`**`()`

Gets sensor possible states.

Returns:

Possible states the sensed property can go through. Currently not supported.

getSensorType()

`public java.lang.Short` **`getSensorType`**`()`

Gets sensor type.

Returns:

Sensor type. Each sensor type is associated with a particular property of a logical/physical device, for example, a voltage sensor might sense/measure/read the voltage through a power supply unit, and a temperature sensor might do similar thing but for the temperature of the unit.

SlotMBean

Declaration

`public interface` **`SlotMBean`** extends `EquipmentHolderMBean`

All Superinterfaces

`EquipmentHolderMBean`

All Known Subinterfaces

`CpciSlotMBean`

Description

This class describes the management interface of the Slot MBean.

Member Summary

Methods

`java.lang.String`	`getAcceptablePlugInUnitTypes()` Gets the acceptable plug-in unit types.
`SlotStatus`	`getSlotStatus()` Gets the slot status.
`java.lang.String`	`getSlotType()` Gets the slot type string.
`javax.management.ObjectName`	`getSoftwareLoad()` Gets the ObjectName of the SoftwareLoad associated with this MBean.
`void`	`setAcceptablePlugInUnitTypes(java.lang.String types)` Sets the acceptable plug-in unit types.
`void`	`setSoftwareLoad(javax.management.ObjectName sw_load)` Sets the ObjectName of the SoftwareLoad associated with this MBean.

Inherited Member Summary
Methods inherited from interface `EquipmentHolderMBean`
`getEquipmentHolderAddress(), getEquipmentHolderLabel(), getEquipmentHolderType()`

Methods

`getAcceptablePlugInUnitTypes()`

`public java.lang.String` **`getAcceptablePlugInUnitTypes`**`()`

Gets the acceptable plug-in unit types.

This attribute identifies the types of plug-in units that can be supported by the slot.

Returns:

The newline separated string representing the acceptable plug-in unit types.

`getSlotStatus()`

`public com.sun.ctmgx.moh.SlotStatus` **`getSlotStatus`**`()`

Gets the slot status.

This attribute provides an indication as to whether or not the slot is empty or full.

Returns:

The slot status value defined as `com.sun.ctmgx.moh.SlotStatus.EMPTY` or `com.sun.ctmgx.moh.SlotStatus.FULL`.

`getSlotType()`

`public java.lang.String` **`getSlotType`**`()`

Gets the slot type string.

This attribute identifies the type of a slot in the chassis, such as a CPCI slot, fan slot, power supply slot, rmm slot, or pmc slot.

Returns:

A String that identifies a type of slot in a chassis.

`getSoftwareLoad()`

`public javax.mNgement.ObjectName` **`getSoftwareLoad`**`()`

Gets the ObjectName of the SoftwareLoad associated with this MBean.

This attribute identifies the software load, if there is any, which is currently designated as the one to be loaded to the plug-in whenever an automatic reload of software is needed.

Returns:

The ObjectName of the SoftwareLoad MBean.

`setAcceptablePlugInUnitTypes(String)`

`public void` **`setAcceptablePlugInUnitTypes`**`(java.lang.String` *types*`)`

Sets the acceptable plug-in unit types.

This attribute identifies the types of plug-in units that can be supported by the slot.

Parameters:

types – A newline separated string representing the acceptable plug-in unit types.

`setSoftwareLoad(ObjectName)`

`public void` **`setSoftwareLoad`**`(javax.management.ObjectName` *sw_load*`)`

Sets the ObjectName of the SoftwareLoad associated with this MBean.

This attribute identifies the software load, if there is any, which is currently designated as the one to be loaded to the plug-in whenever an automatic reload of software is needed.

Returns:

The ObjectName of the SoftwareLoad MBean.

SoftwareMonitorMBean

Declaration

public interface **SoftwareMonitorMBean**.

Member Summary	
Fields	
`static java.lang.String`	`ETHER_STATS_SERVICE` Ethernet Driver Statistic service name.
`static java.lang.String`	`ETHER_STATS_SERVICE_OBJ` Ethernet Driver Statistic service object name.

Member Summary *(Continued)*

`static java.lang.String`	`IP_SERVICE` Internet Protocol service name.
`static java.lang.String`	`IP_SERVICE_OBJ` Internet Protocol service object name.
`static java.lang.String`	`NFS_SERVICE` Network File System service name.
`static java.lang.String`	`NFS_SERVICE_OBJ` Network File System service object name.
`static java.lang.String`	`PLATFORM_MGT_SERVICE` Platform Management service name.
`static java.lang.String`	`PLATFORM_MGT_SERVICE_OBJ` Platform Management service object name.
`static java.lang.String`	`TCP_SERVICE` Transport Control Protocol service name.
`static java.lang.String`	`TCP_SERVICE_OBJ` Transport Control Protocol service object name.
`static java.lang.String`	`TFTP_SERVICE` Trivial File Transfer Protocol service name.
`static java.lang.String`	`TFTP_SERVICE_OBJ` Trivial File Transfer Protocol service object name.
`static java.lang.String`	`UDP_SERVICE` User Datagram Protocol service name.
`static java.lang.String`	`UDP_SERVICE_OBJ` User Datagram Protocol service object name.
`static java.lang.String`	`UFS_SERVICE` Unix File System service name.
`static java.lang.String`	`UFS_SERVICE_0BJ` Unix File System service object name.
Methods	
`java.util.Set`	`getSoftwareServiceList()` Gets a list of object names representing the software services.

Fields

`ETHER_STATS_SERVICE`

```
public static final java.lang.String ETHER_STATS_SERVICE
```

Ethernet Driver Statistic service name.

`ETHER_STATS_SERVICE_OBJ`

```
public static final java.lang.String ETHER_STATS_SERVICE_OBJ
```

Ethernet Driver Statistic service object name.

`IP_SERVICE`

```
public static final java.lang.String IP_SERVICE
```

Internet Protocol service name.

`IP_SERVICE_OBJ`

```
public static final java.lang.String IP_SERVICE_OBJ
```

Internet Protocol service object name.

`NFS_SERVICE`

```
public static final java.lang.String NFS_SERVICE
```

Network File System service name.

`NFS_SERVICE_OBJ`

```
public static final java.lang.String NFS_SERVICE_OBJ
```

Network File System service object name.

`PLATFORM_MGT_SERVICE`

```
public static final java.lang.String PLATFORM_MGT_SERVICE
```

Platform Management service name. It is PICL on Solaris or SMF on Alarm Card.

`PLATFORM_MGT_SERVICE_OBJ`

```
public static final java.lang.String PLATFORM_MGT_SERVICE_OBJ
```

Platform Management service object name.

`TCP_SERVICE`

```
public static final java.lang.String TCP_SERVICE
```

Transport Control Protocol service name.

TCP_SERVICE_OBJ

`public static final java.lang.String TCP_SERVICE_OBJ`

Transport Control Protocol service object name.

TFTP_SERVICE

`public static final java.lang.String TFTP_SERVICE`

Trivial File Transfer Protocol service name.

TFTP_SERVICE_OBJ

`public static final java.lang.String TFTP_SERVICE_OBJ`

Trivial File Transfer Protocol service object name.

UDP_SERVICE

`public static final java.lang.String UDP_SERVICE`

User Datagram Protocol service name.

UDP_SERVICE_OBJ

`public static final java.lang.String UDP_SERVICE_OBJ`

User Datagram Protocol service object name.

UFS_SERVICE

`public static final java.lang.String UFS_SERVICE`

Unix File System service name.

UFS_SERVICE_OBJ

`public static final java.lang.String UFS_SERVICE_OBJ`

Unix File System service object name.

Methods

getSoftwareServiceList()

`public java.util.Set` **`getSoftwareServiceList`**`()`

Gets a list of object names representing the software services.

Returns:

Set of objects of the software services. Each object contains a string representing one of the following the object names:

```
com.sun.ctmgx.moh.SoftwareMonitorMbean.NFS_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.UFS_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.TCP_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.UDP_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.IP_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.TFTP_SERVICE_OBJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.ETHER_STATS_SERVICE_O
BJ
com.sun.ctmgx.moh.SoftwareMonitorMbean.PLATFORM_MGT_SERVICE_
OBJ
```

SoftwareServiceMBean

Declaration

`public interface` **`SoftwareServiceMBean`**

All Known Subinterfaces:

```
CgtpServiceMBean, EtherIfStatsMBean,IpServiceMBean,
NfsServiceMBean, RnfsServiceMBean, TcpServiceMBean,
UdpServiceMBean,UfsServiceMBean
```

Description

This class describes the interface of Software Service MBean.

This is the base class from which all software services are derived.

Software services in the systems such as NFS, NIS, CGTP, etc. are modeled via MOH software modules. Each software module represents a service.

The software modules of MOH running on host CPU and satellite CPU monitor the software services in the system. Software services can be a software sub-system such as network stacks (TCP, UDP, IP), or I/O drivers such as Ethernet drivers, or processes or daemons such as NFS, UFS, TFTP, reliable NFS, etc.

In general, a service contains two parts: The daemon monitor part and the statistic part. The daemon monitor part deals with the installed/not installed, and running/not running status of a daemon. The statistic part deals with error count, error thresholds, polling, etc. Not all of the services contain two parts, since some services might not be implemented as daemons.

Member Summary

Methods

`javax.management.ObjectName[]`	`getDaemonList()` Get the list of daemons supporting the service.
`java.lang.String`	`getName()` Gets the name of the service.
`java.lang.Integer`	`getNumExcessiveIntervals()` Gets configured number of polling intervals that exceed the max threshhold
`java.lang.Integer`	`getPollingInterval()` Gets polling interval.
`java.lang.String`	`getStatus()` Gets the status of the service.
`void`	`setNumExcessiveIntervals(java.lang.Integer numExcessiveIntervals)` Sets number of polling intervals that exceed the max threshold or number of excessive intervals that the service exceeds error threshold.
`void`	`setPollingInterval(java.lang.Integer milliSecs)` Sets polling interval in milliseconds.
`void`	`startPolling()` Starts polling for exceeding thresholds and mount failure notifications.
`void`	`stopPolling()` Stops polling.

Methods

`getDaemonList()`

`public javax.management.ObjectName() **getDaemonList**()`

Gets the list of daemons supporting the service.

Returns:

The list of object name of DaemonMbean(s). Each object name is defined as `com.sun.ctmgx.moh.SoftwareMonitorMbean."softwareservice name"."daemon name"`

`getName()`

`public java.lang.String` **getName()**

Gets the name of the service. All of available services are defined in this method.

Returns:

The name of the service defined as one of the following:

`com.sun.ctmgx.moh.SoftwareMonitorMbean.NFS_SERVICE`
com.sun.ctmgx.moh.SoftwareMonitorMbean.UFS_SERVICE
com.sun.ctmgx.moh.SoftwareMonitorMbean.TCP_SERVICE
`com.sun.ctmgx.moh.SoftwareMonitorMbean.UDP_SERVICE`
com.sun.ctmgx.moh.SoftwareMonitorMbean.IP_SERVICE
`com.sun.ctmgx.moh.SoftwareMonitorMbean.TFTP_SERVICE`
com.sun.ctmgx.moh.SoftwareMonitorMbean.ETHER_STATS_SERVICE
or `string("") if there is none`.

`getNumExcessiveIntervals()`

`public java.lang.Integer` **`getNumExcessiveIntervals`**`()`

Gets the number of excessive intervals that the service exceeds error threshold.

This excessive polling interval threshold indicates the maximum number of consecutive polling intervals that the corresponding service exceeds the maximum allowable threshold before an error event is generated.

Returns:

The number of excessive intervals.

`getPollingInterval()`

`public java.lang.Integer` **`getPollingInterval`**`()`

Gets polling interval of the service.

The polling interval indicates how often the service is polled for status.

Returns:

The polling interval in millisecond. The default is 15 minutes.

`getStatus()`

`public java.lang.String` **`getStatus`**`()`

Gets status of the service.

Returns:

The status of the service defined as `string("up")`, or `string("down")`.

`setNumExcessiveIntervals(Integer)`

`public void` **`setNumExcessiveIntervals`**`(java.lang.Integer` *numExcessiveIntervals*`)`

Sets the number of configured polling intervals threshold or number of excessive intervals that the service exceeds error threshold.

This excessive polling interval threshold indicates the maximum number of consecutive polling intervals that the corresponding service exceeds the maximum allowable threshold before an error event is generated.

Parameters:

numExcessiveIntervals – interval threshold to set. The default of this threshold is 1.

`setPollingInterval(Integer)`

`public void` **`setPollingInterval`**`(java.lang.Integer` *milliSecs*`)`

Sets polling interval of the service. The polling interval indicates how often the service is polled for status.

Parameters:

milliSecs– polling interval to set. The default is 15 minutes. The minimum interval is 15 seconds.

`startPolling()`

`public void` **`startPolling`**`()`

Start polling the service.

`stopPolling()`

`public void` **`stopPolling`**`()`

Stop polling the service.

TcpServiceMBean

Declaration

`public interface` **`TcpServiceMBean`** extends `SoftwareServiceMBean`

Description

This class describes the interface of Unix File System (UFS) service.

Member Summary	
Methods	
`java.lang.Float`	`getCurrntInErrorThreshold()` Gets the current threshold of segments received in error.
`java.lang.Float`	`getCurrntOutRstsThreshold()` Gets the current threshold of segments sent with RST flag.
`java.lang.Float`	`getMaxInErrorThreshold()` Gets the maximum threshold of segments received in error.
`java.lang.Float`	`getMaxOutRstsrThreshold()` Gets the maximum threshold of segments sent with RST flag.
`void`	`setMaxInErrorThreshold(java.lang.Float newThreshold)` Sets the maximum threshold of segments received in error.
`void`	`setMaxOutRstsrThreshold(java.lang.Float newThreshold)` Sets the maximum threshold of segments sent with RST flag.

Inherited Member Summary
Methods inherited from interface `SoftwareServiceMBean`
`getDaemonList(), getName(), getNumExcessiveIntervals(),getPollingInterval(), getStatus(),setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling(), stopPolling()`

Methods

`getCurrentInErrorThreshold()`

`public java.lang.Float` **getCurrentInErrorThreshold()**

Gets the current threshold of segments received in error.

Returns:

The threshold in percentage of number of errors over the total received segments of the current polling interval.

`getCurrentOutRstsThreshold()`

`public java.lang.Float` **getCurrentOutRstsThreshold()**

Gets the current threshold of segments sent with RST flag.

Returns:

The threshold in percentage of number of errors over the total sent segments of the current polling interval.

`getMaxInErrorThreshold()`

`public java.lang.Float` **getMaxInErrorThreshold()**

Gets the maximum threshold of segments received in error.I

Returns:

The configured threshold in percentage of number of errors over the total received segments of the current polling interval.

`getMaxOutRstsThreshold()`

`public java.lang.Float` **getMaxOutRstsThreshold()**

Gets the maximum threshold of segments sent with RST flag.

Returns:

The max threshold in percentage of number of errors over the total received segments of the current polling interval.

`setMaxInErrorThreshold(Float)`

`public void` **setMaxInErrorThreshold**`(java.lang.Float` *newThreshold*`)`

Sets the maximum threshold of segments received in error. If the threshold exceeds a number of configured consecutive intervals, a notification of class AttributeChangeNotification is sent to the clients for the change of attribute “CurrentInErrorThreshold”.

Parameters:

newThreshold – maximum threshold to set.

`setMaxOutRstsThreshold(Float)`

`public void` **setMaxOutRstsThreshold**`(java.lang.Float` *newThreshold*`)`

Sets the maximum threshold of segments sent with RST flag. If the threshold exceeds a number of configured consecutive intervals, a notification of class AttributeChangeNotification is sent to the clients for the change of attribute CurrentOutRstsThreshold.

Parameters:

newThreshold – maximum threshold to set.

TerminationPointMBean

Declaration

`public interface` **TerminationPointMBean**

Description

This class describes the management interface of the Termination Point MBean.

Member Summary

Methods

`AdministrativeState`	`getAdministrativeState()` Gets the administrative state.
`javax.management.ObjectName`	`getAlarmSeverityProfilePointer()` Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.
`OperationalState`	`getOperationalState()` Gets the operational state.
`java.lang.String`	`getPhysicalPathType()` Gets the physical path type.
`int`	`getPortID()` Gets the port ID.
`java.lang.String`	`getPortLabel()` Gets the port label.
`void`	`setAdministrativeState(AdministrativeState admin_state)` Sets the administrative state.
`void`	`setAlarmSeverityProfilePointer(ObjectName asp)` Sets the AlarmSeverityProfile associated with this MBean.

Methods

getAdministrativeState()

`public com.sun.ctmgx.moh.AdministrativeState` **`getAdministrativeState`**`()`

Gets the administrative state.

This attribute is used to activate (unlocked) and deactivate (locked) the function performed by the physical path termination point.

Returns:

The administrative state value defined as either

`com.sun.ctmgx.moh.AdministrativeState.LOCKED` or
`com.sun.ctmgx.moh.AdministrativeState.UNLOCKED`

getAlarmSeverityProfilePointer()

```
public javax.management.ObjectName
getAlarmSeverityProfilePointer()
```

Gets the ObjectName of the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignmentsare used.

Returns:

The ObjectName of the AlarmSeverityProfile MBean.

getOperationalState()

```
public com.sun.ctmgx.moh.OperationalState getOperationalState()
```

Gets the operational state.

This attribute identifies whether or not the physical path termination point is capable of performing its normal functions, that is, enabled or disabled.

Returns:

One of the following operational state values:

```
com.sun.ctmgx.moh.OperationalState.ENABLED
com.sun.ctmgx.moh.OperationalState.DISABLED
com.sun.ctmgx.moh.OperationalState.UNKNOWN
```

getPhysicalPathType()

```
public java.lang.String getPhysicalPathType()
```

Gets the physical path type.

This attribute identifies the type of physical path being terminated (for example DS1, DS3, SONET STS-3c, or Ethernet).

Returns:

The physical path type.

`getPortID()`

`public int` **`getPortID`**`()`

Gets the port ID.

This attribute identifies the port on the line card where the physical path terminates.

Returns:

The port ID.

`getPortLabel()`

`public java.lang.String` **`getPortLabel`**`()`

Gets the port label.

This attribute provides the external label string of this physical path termination point object.

Returns:

The port label.

`setAdministrativeState(AdministrativeState)`

`public void`
`setAdministrativeState``(com.sun.ctmgx.moh.AdministrativeState` *admin_state)*

Sets the administrative state.

This attribute is used to activate (unlocked) and deactivate (locked) the function performed by the physical path termination point.

Parameters:

admin_state – The administrative state value defined as either:

```
com.sun.ctmgx.moh.AdministrativeState.LOCKED
com.sun.ctmgx.moh.AdministrativeState.UNLOCKED
```

`setAlarmSeverityProfilePointer(ObjectName)`

`public void` **`setAlarmSeverityProfilePointer`**`(ObjectName` *asp*`)`

Sets the ObjectName of the AlarmSeverityProfile associated with this MBean.

This attribute provides a pointer to the instance of the Alarm Severity Assignment Profile MBean that contains the severity assignments for the alarms reported by this MBean. When the value of this attribute is set to `null`, default severity assignments are used.

Parameters:

asp – The ObjectName of the AlarmSeverityProfile MBean.

UdpServiceMBean

Declaration

`public interface` **`UdpServiceMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the interface of Unix File System (UFS) service.

Member Summary

Methods

`java.lang.Float`	`getCurrentInErrorThreshold()` Gets the current threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
`java.lang.Float`	`setMaxInErrorThreshold` Gets the maximum threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
`void`	`setMaxInErrorThreshold(java.lang.Float newThreshold)` Sets the maximum threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

`getDaemonList(), getName(), getNumExcessiveIntervals(), getPollingInterval(), getStatus(), setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling(), stopPolling()`

Methods

getCurrentInErrorThreshold()

`public java.lang.Float` **`getCurrentInErrorThreshold()`**

Gets the current threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

Returns:

The threshold in percentage of number of errors over the total receiving datagrams of the current polling interval.

getMaxInErrorThreshold()

`public java.lang.Float` **getMaxInErrorThreshold()**

Gets the maximum threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

Returns:

The maximum threshold in percentage.

setMaxInErrorThreshold(Float)

`public void` **setMaxInErrorThreshold**`(java.lang.Float` *newThreshold*`)`

Sets the maximum threshold of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

Parameters:

newThreshold – maximum threshold to set.

UfsServiceMBean

Declaration

`public interface` **`UfsServiceMBean`** extends `SoftwareServiceMBean`

All Superinterfaces

`SoftwareServiceMBean`

Description

This class describes the interface of UfsService class.

Member Summary	
Methods	
`java.lang.String[]`	`getExcessiveThreshHoldFileSystemList()` Gets the list of file systems exceeding threshold.
`java.lang.Float`	`getFileSystemMaxThreshHold()` Gets the threshold percentage usage of the file system.
`void`	`setFileSystemMaxThreshHold(java.lang.Float threshHold)` Sets the threshold percentage usage of the file system.

Inherited Member Summary

Methods inherited from interface `SoftwareServiceMBean`

`getDaemonList(),getName(), getNumExcessiveIntervals(), getPollingInterval(), getStatus(), setNumExcessiveIntervals(Integer), setPollingInterval(Integer), startPolling(), stopPolling()`

Methods

`getExcessiveThreshHoldFileSystemList()`

`public java.lang.String[] getExcessiveThreshHoldFileSystemList()`

Gets the list of file systems exceeding threshold.

Returns:

List of file system.

Null is returned if there is no file systems exceeding threshold.

`getFileSystemMaxThreshHold()`

`public java.lang.Float getFileSystemMaxThreshHold()`

Gets the threshold percentage usage of the file system.

Returns:

Usage in percentage.

`setFileSystemMaxThreshHold(Float)`

`public void setFileSystemMaxThreshHold(java.lang.Float` *threshHold*`)`

Sets the threshold percentage usage of the file system.

Parameters:

threshHold–maximum threshold to set.

Netra CT Management Agent Class Descriptions

This section contains descriptions of the following management agent classes:

AdministrativeState

Declaration

```
public class AdministrativeState extends java.lang.Object
implements java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.AdministrativeState
```

All Implemented Interfaces:

`java.io.Serializable`

Description

This class defines the administrative states of an object. Currently, for Netra CT 410, and CT 810, there is no implementation in MOH to perform lockout processing based on this state.

Member Summary	
Fields	
static AdministrativeState	LOCKED
static AdministrativeState	UNLOCKED
Methods	
Boolean	equals(java.lang.Object obj)
int	hashCode()
int	intValue()
java.lang.String	toString()

Inherited Member Summary

Methods inherited from class `Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

`LOCKED`

`public static final com.sun.ctmgx.moh.`**`AdministrativeState`**
`LOCKED`

Indicates the related object is in-use or busy performing a function.

`UNLOCKED`

`public static final com.sun.ctmgx.moh.`**`AdministrativeState`**
`UNLOCKED`

Indicates the related object is not in-use or not busy performing a function.

Methods

`equals(Object)`

`public Boolean` **`equals`**`(java.lang.Object obj)`

Overrides:

`equals` in class `Object`

`hashCode()`

`public int` **`hashCode`**()

Overrides:

`hashCode` in class `Object`

`intValue()`

`public int` **`intValue`**`()`

`toString()`

`public java.lang.String` **`toString`**()

Overrides:

`toString` in class `Object`

AlarmNotification

Declaration

```
public class AlarmNotification extends
com.sun.ctmgx.moh.AlarmNotification
```

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--javax.management.Notification
              |
               +--com.sun.ctmgx.moh.AlarmNotification
```

Description

The Alarm Notification class represents an alarm notification emitted by an MBean. This notification is used to notify the management system when a failure has been detected or cleared.

It contains a reference to the source MBean: if the notification has been forwarded through the MBean server, this is the object name of the MBean. If the listener has registered directly with the MBean, this is a direct reference to the MBean.

Note – To receive alarms, the alarm listener must: (1) Assign an alarm severity profile to an object which is in the list of alarm types (for example, fan, CPU thermistor, or CPU card memory monitor) (2) Enable alarm types.

Member Summary	
Constructors	
	`AlarmNotification(AlarmType alarmType, java.lang.Object source, long sequenceNumber, java.lang.String message, AlarmSeverity perceivedSeverity, java.lang.String specificProblems, Boolean backedUpStatus, javax.management.ObjectName backUpObject, java.lang.String proposedRepairActions, java.util.Set failedSwitchComponentList)` Constructs an alarm notification object.
Methods	
`AlarmType`	`getAlarmType()` Gets the alarm type.

Member Summary *(Continued)*	
`Boolean`	`getBackedUpStatus()` Gets the backed-up status.
`javax.management.ObjectName`	`getBackUpObject()` Gets the back-up object.
`java.util.Set`	`getFailedSwitchComponentList()` Gets the list of failed switch components.
`AlarmSeverity`	`getPerceivedSeverity()` Gets the perceived severity of the alarm.
`java.lang.String`	`getProposedRepairActions()` Gets the proposed repair actions.
`java.lang.String`	`getSpecificProblems()` Gets the specific problems of the alarm.

Inherited Member Summary

Methods inherited from class `javax.management.Notification`

`getMessage, getSequenceNumber, getSource, getTimeStamp, getType, getUserData, setSequenceNumber, setSource, setTimeStamp, setUserData`

Methods inherited from class `java.util.EventObject`

`toString`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

`AlarmNotification`

`public` **`AlarmNotification`**`(com.sun.ctmgx.moh.AlarmType` *alarmType*, `java.lang.Object` *source*, `long` *sequenceNumber*, `java.lang.String` *message*, `com.sun.ctmgx.moh.AlarmSeverity` *perceivedSeverity*, `java.lang.String` *specificProblems*, `Boolean` *backedUpStatus*, `java.management.ObjectName` *backUpObject*, `java.lang.String` *proposedRepairActions*, `java.util.Set` *failedSwitchComponentList*`)`

Constructs an alarm notification object.

Parameters:

alarmType – one of the predefined AlarmType instances.

source – The notification source, that is, the MBean that emits the notification.

sequenceNumber – The notification sequence number within the source object.

message – A String containing the message of the notification.

specificProblems – Indicates further refinements to the problem identified by the alarm type.

perceivedSeverity – One of the perceived severity values in the `com.sun.ctmgx.moh.AlarmSeverity` class.

backedUpStatus – A Boolean indication as to whether or not the failed entity has been backed-up.

backUpObject – The ObjectName of the MBean providing back-up services to the failed entity.

proposedRepairActions – Indicates proposed repair actions for the problem identified by the alarm.

failedSwitchComponentList – A set of failed (or possibly failed) components associated with this alarm identified by their ObjectName.

Methods

`getAlarmType()`

`public com.sun.ctmgx.moh.AlarmType` **`getAlarmType`**`()`

Gets the alarm type.

Returns:

AlarmType enumeration.

`getBackedUpStatus()`

`public Boolean` **`getBackedUpStatus`**`()`

Gets the backed-up status.

If the value of this object is true, the agent reported in this notification that the failed object had been backed up.

Returns:

The backed-up status.

`getBackUpObject()`

`public javax.management.ObjectName` **`getBackUpObject`**`()`

Gets the back-up object.

Indicates the object that provided back-up services to the failed object.

Returns:

The ObjectName of the back-up object.

`getFailedSwitchComponentList()`

`public java.util.Set` **`getFailedSwitchComponentList`**`()`

Gets the list of failed switch components.

Returns:

The list of failed switch components.

`getPerceivedSeverity()`

`public com.sun.ctmgx.moh.AlarmSeverity` **`getPerceivedSeverity`**`()`

Gets the perceived severity of the alarm.

Returns:

One of the constants for perceived severity defined in the class `com.sun.ctmgx.moh.AlarmSeverity`.

`getProposedRepairActions()`

`public java.lang.String` **`getProposedRepairActions`**`()`

Gets the proposed repair actions.

Indicates proposed repair actions reported by the agent for the problem identified by the alarm. If more than one action is described in this object, the problem descriptions are separated by newline characters.

Returns:

The proposed repair actions.

`getSpecificProblems()`

`public java.lang.String` **`getSpecificProblems`**`()`

Gets the specific problems of the alarm.

Indicates further refinements to the problem identified by the alarm type. If more than one specific problem is described in this object, the problem descriptions are separated by newline characters.

Returns:

The specific problems of the alarm.

AlarmNotificationFilter

Declaration

```
public class AlarmNotificationFilter implements
java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.AlarmNotificationFilter
```

All Implemented Interfaces:

`javax.management.NotificationFilter,java.io.Serializable`

Description

This class describes the filtering of AlarmNotification notifications by selecting the types and severities of interest.

It manages a list of enabled types and severities for which notifications should be sent. Methods in this class allow users to enable/disable (allow/disallow of notifications based on severities and types) as many types and severities as required.

Member Summary	
Constructors	
	`AlarmNotificationFilter()`
Methods	
`void`	`disableAlarmType(AlarmType type)` Disables a specific alarm type.
`void`	`disableAllAlarmTypes()` Disables all alarm types.
`void`	`disableAllSeverities()` Disables all the severities that were enabled.
`void`	`disableSeverity(AlarmSeverity severity)` Disables a specific severity type. Removes a severity for which you are no longer interested in receiving notifications.
`void`	`enableAlarmType(AlarmType type)` Enables a specific type.
`void`	`enableAllAlarmTypes()` Enables all alarm types.

Member Summary *(Continued)*

`void`	`enableSeverity(AlarmSeverity severity)` Adds a severity for which you are interested in receiving notifications.
`java.util.Vector`	`getEnabledAlarmTypes()` Gets the alarm types currently enabled.
`java.util.Vector`	`getEnabledSeverities()` Gets the list of enabled severities for this filter.
`Boolean`	`isNotificationEnabled(javax.management.Notification notification)` Queries whether notification is needed.

Inherited Member Summary

Methods inherited from class `java.lang.Object`

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

AlarmNotificationFilter()

`public AlarmNotificationFilter()`

Methods

disableAlarmType(AlarmType)

`public void disableAlarmType(com.sun.ctmgx.moh.AlarmType type)`

Disables a specific alarm type. To disable a specific Alarm type. If all alarm types are enabled through an enableAllAlarmTypes() call, this call will not have any effect.

disableAllAlarmTypes()

`public void disableAllAlarmTypes()`

Disables all alarm types. This method disables all enabled Alarm types as well as the effect of the enableAllAlarmTypes() call.

disableAllSeverities()

`public void disableAllSeverities()`

Disables all the severities that were enabled.

`disableSeverity(AlarmSeverity)`

`public void` **`disableSeverity`**(com.sun.ctmgx.moh.AlarmSeverity *severity*)

Disables a specific severity type. Remove a severity for which you are no longer interested in receiving notifications.

Parameters:

severity – The severity value.

`enableAlarmType(AlarmType)`

`public void` **`enableAlarmType`**`(com.sun.ctmgx.moh.AlarmType` *type*`)`

This enables a specific Alarm type. If all Alarm types are already enabled through the enableAllAlarmTypes() call, then type is also already enabled. In this case, to enable just a specific Alarm type, disableAllAlarmTypes() call has to be performed first and then this method can be invoked.

`enableAllAlarmTypes()`

`public void` **`enableAllAlarmTypes`**`()`

Enables all alarm types.This will enable all kinds of Alarm. This call enables the notifications with the type string that have a prefix of AlarmType.ALARM_TYPE_PREFIX, for example `netract.moh.alarm`.

`enableSeverity(AlarmSeverity)`

`public void` **`enableSeverity`**(`com.sun.ctmgx.moh.AlarmSeverity` **`severity`**)
throws IllegalArgumentException

Adds a severity for which you are interested in receiving notifications.

Parameters:

severity – The severity value.

Throws:

`java.lang.IllegalArgumentException`

`getEnabledAlarmTypes()`

`public java.util.Vector` **`getEnabledAlarmTypes`**`()`

Gets the alarm types currently enabled.

Returns:

A vector of strings representing alarm types.

`getEnabledSeverities()`

`public java.util.Vector` **`getEnabledSeverities`**`()`

Gets the list of enabled severities for this filter.

Returns:

The list containing the severities for which notifications should be sent.

`isNotificationEnabled(Notification)`

`public Boolean`
`isNotificationEnabled``(javax.management.Notification` *`notification`*`)`

Queries whether notification is needed. This method is invoked before sending the specified notification to the listener.

This filter compares the type and severity of the specified alarm notification with each enabled type and severity. If the type equals one of the enabled types and the severity equals one of the enabled severities, the notification must be sent to the listener so this method returns `true`.

Parameters:

notification – The alarm notification to be sent.

Returns:

`true` if the notification has to be sent to the listener, `false` otherwise.

AlarmSeverity

Declaration

```
public class AlarmSeverity implements java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.AlarmSeverity
```

All Implemented Interfaces:

`java.io.Serializable`

Description

This class defines the alarm severity objects for use with alarm notification.

Member Summary

Fields

`static AlarmSeverity`	`CLEARED` Indicates that the alarm condition is clear.
`static AlarmSeverity`	`CRITICAL` Indicates that the alarm is critical.
`static AlarmSeverity`	`INDETERMINATE` Indicates that the alarm condition is indeterminate.
`static AlarmSeverity`	`MAJOR` Indicates that the alarm is major.
`static AlarmSeverity`	`MINOR` Indicates that the alarm is minor.
`static AlarmSeverity`	`WARNING` Indicates that the alarm is minor.

Methods

`Boolean`	`equals(java.lang.Object obj)`
`int`	`hashCode()`
`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `java.lang.Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

CLEARED

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`CLEARED`**

Indicates the alarm condition is clear.

CRITICAL

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`CRITICAL`**

Indicates the alarm is critical.

INDETERMINATE

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`INDETERMINATE`**

Indicates the alarm condition is indeterminate.

MAJOR

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`MAJOR`**

Indicates the alarm is major.

MINOR

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`MINOR`**

Indicates the alarm is minor.

WARNING

`public static final com.sun.ctmgx.moh.AlarmSeverity` **`WARNING`**

Indicates the alarm is a warning

Methods

equals(Object)

`public Boolean` **`equals`**`(java.lang.Object obj)`

Overrides:

`equals` in class `Object`

hashCode()

`public int` **`hashCode`**`()`

Overrides:

`hashCode` in class `Object`

intValue()

`public int` **`intValue`**`()`

`toString()`

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `Object`

AlarmType

Declaration

`public class` **`AlarmType`** `extends jaba.lang.Object implements java.io.Serializable`

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.AlarmType
```

All Implemented Interfaces:

`java.io.Serializable`

Description

This class is an enumeration of predefined Alarm types; the user needs to use one of the predefined types to construct an AlarmNotification object.

Member Summary

Fields	
`static AlarmType`	`FAN_FAILURE` Alarm type that indicates a fan failure alarm.
`static AlarmType`	`FUSE_FAILURE` Alarm type that indicates a fuse failure alarm.
`static AlarmType`	`HIGH_MEMORY_UTILIZATION` Alarm type that indicates a high physical memory utilization alarm.
`static AlarmType`	`HIGH_TEMPERATURE` Alarm type that indicates a high temperature alarm.
Methods	
`Boolean`	`equals(java.lang.Object obj)`
`int`	`hashCode()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `java.lang.Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

FAN_FAILURE

`public static final com.sun.ctmgx.moh.AlarmType` **`FAN_FAILURE`**

Alarm type that indicates a fan failure alarm.

FUSE_FAILURE

`public static final com.sun.ctmgx.moh.AlarmType` **`FUSE_FAILURE`**

Alarm type that indicates a fuse failure alarm.

HIGH_MEMORY_UTILIZATION

`public static final com.sun.ctmgx.moh.AlarmType`
`HIGH_MEMORY_UTILIZATION`

Alarm type that indicates a high physical memory utilization alarm.

HIGH_TEMPERATURE

`public static final com.sun.ctmgx.moh.AlarmType` **`HIGH_TEMPERATURE`**

Alarm type that indicates a high temperature alarm.

Methods

equals(Object)

`public Boolean` **`equals`**`(java.lang.Object obj)`

Overrides:

`equals` in class `Object`.

hashCode()

`public int` **hashCode()**

Overrides:

`hashCode` in class `Object`.

`intValue()`

`public int` **intValue()**

`toString()`

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `Object`.

AttributeChangeNotification

Declaration

```
public class AttributeChangeNotification extends
javax.management.AttributeChangeNotification
```

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--javax.management.Notification
              |
              +--javax.management.AttributeChangeNotification
                    |
                    +--com.sun.ctmgx.moh.AttributeChangeNotification
```

All Implemented Interfaces

`java.io.Serializable`

Description

This class defines definitions of the attribute change notifications sent by MBeans.

It is up to the MBean owning the attribute of interest to create and send attribute change notifications when the attribute change occurs, so the `NotificationBroadcaster` interface has to be implemented by any MBean interested in sending attribute change notifications.

Member Summary

Fields	
`static java.lang.String`	`ATTRIBUTE_CHANGE` Notification type that indicates that the observed MBean attribute value has changed.
Constructors	
`AttributeChangeNotification`	`(java.lang.Object source, long sequenceNumber, java.lang.String message, java.lang.String attributeName, java.lang.String attributeType, java.lang.Object oldValue, java.lang.Object newValue)` Constructs an attribute change notification object whose type string is `netract.moh.attribute.change`. In addition to the information common to all notifications, the caller must supply the name and type of the attribute, as well as its old and new values.
Methods	
`java.lang.String`	`getType()` Gets type of notification.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

`ATTRIBUTE_CHANGE`

`public static final java.lang.String` **`ATTRIBUTE_CHANGE`**

Notification type that indicates that the observed MBean attribute value has changed.

The value of this type string is `netract.moh.attribute.change`.

Constructors

`AttributeChangeNotification`

`public` **`AttributeChangeNotification`**`(java.lang.Object` *source*`, long` *sequenceNumber*`, java.lang.String` *message*`, java.lang.String` *attributeName*`, java.lang.String` *attributeType*`, java.lang.Object` *oldValue*`, java.lang.Object` *newValue*`)`

Construct an attribute change notification object whose type string is netract.moh.attribute.change. In addition to the information common to all notifications, the caller must supply the name and type of the attribute, as well as its old and new values.

Parameters:

source – The notification source, that is, the MBean that emits the notification.

sequenceNumber – The notification sequence number within the source object.

message – A String containing the message of the notification.

attributeName – A String specifying the name of the attribute.

attributeType – A String specifying the type of the attribute.

oldValue – An object representing the value of the attribute before the change.

newValue – An object representing the value of the attribute after the change.

Methods

`getType()`

`public java.lang.String` **`getType`**()

Gets the notification type.

Overrides:

getType in class javax.management.Notification

Returns:

ATTRIBUTE_CHANGE. The method of `javax.management.Notification` is overridden to return ATTRIBUTE_CHANGE.

AttributeChangeNotificationFilter

Declaration

```
public class AttributeChangeNotificationFilter extends
javax.management.AttributeChangeNotificationFilter
```

```
java.lang.Object
  |
  +--javax.management.AttributeChangeNotificationFilter
        |
         +--com.sun.ctmgx.moh.AttributeChangeNotificationFilter
```

All Implemented Interfaces

`javax.management.NotificationFilter, java.io.Serializable`

Description

This class describes the filtering performed on the name of the observed attribute.

It manages a list of enabled attributes for which notifications should be sent when their attribute changes. A method in this class allows users to decide whether notifications should be sent for attribute changes.

Member Summary	
Constructors	
	`AttributeChangeNotificationFilter()`
Methods	
`Boolean`	`isNotificationEnabled(Notification notification)` This method is invoked before sending the specified notification to the listener.

Inherited Member Summary

Methods inherited from class javax.management.AttributeChangeNotificationFilter

`disableAllAttributes, disableAttribute, enableAttribute, getEnabledAttributes`

Methods inherited from class java.lang.Object

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

`AttributeChangeNotificationFilter`

`public` **`AttributeChangeNotificationFilter`**`()`

Methods

`isNotificationEnabled(Notification)`

`public Boolean`
`isNotificationEnabled``(javax.management.Notification` *notification*`)`

This method is invoked before sending the specified notification to the listener.

This filter compares the attribute name of the specified attribute change notification with each enabled attribute name. If the attribute name equals one of the enabled attribute names, the notification must be sent to the listener so this method returns `true`.

Parameters:

notification – The attribute change notification to be sent.

Returns:

`true` if the notification has to be sent to the listener, `false` otherwise.

AuthClient

Declaration

```
public class AuthClient extends java.lang.Object
```

```
java.lang.Object
   |
   +--com.sun.ctmgx.moh.AuthClient
```

Description

This class defines the client utility routines, particularly for authentication.

Member Summary

Constructors	
	`AuthClient()`
Methods	
`static void`	setAuthInfo(com.sun.jdmk.comm.RmiConnectorClient client, java.lang.String user, java.lang.String passwd) Set the authentication information for a Connector client.

Inherited Member Summary

Methods inherited from class `java.lang.Object`

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor

`AuthClient`

`public` **`AuthClient()`**

Method

`setAuthInfo`

`public static void` **`setAuthInfo`**`(com.sun.jdmk.comm.RmiConnectorClient` *client*`, java.lang.String` *user*`, java.lang.String` *passwd*`)`

Set the authentication information for a Connector client.

Parameters:

client – an instance of RmiConnectorClient.

user – The username which is the same as the user configured on the Alarm card.

passwd – The password as configured on the Alarm card for the username.

AvailabilityStatus

Declaration

```
public class AvailabilityStatus extends java.lang.Object
implements java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.AvailabilityStatus
```

All Implemented Interfaces:

`java.io.Serializable`

Description

This class defines the availability status for a plug-in unit object.

Member Summary	
Fields	
`static AvailabilityStatus`	`AVAILABLE` This class defines the availability status for a plug-in unit object.
`static AvailabilityStatus`	`DEPENDENCY` Indicates the object is in dependency state.
`static AvailabilityStatus`	`FAILED` Indicates the object is in failed state.
`static AvailabilityStatus`	`INTEST` Indicates the object is under test.
`static AvailabilityStatus`	`NOTINSTALLED` Indicates the object is not properly configured.
`static AvailabilityStatus`	`OFFLINE` Indicates the object is in offline state.
`static AvailabilityStatus`	`POWEROFF` Indicates the object is in power off state.
`static AvailabilityStatus`	`UNKNOWN` Indicates the state of the object is unknown.
Methods	
`Boolean`	`equals(java.lang.Object obj)`

Member Summary *(Continued)*	
`int`	`hashCode()`
`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `java.lang.Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

`AVAILABLE`

`public static final com.sun.ctmgx.moh.AvailabilityStatus` **`AVAILABLE`**

Indicates the object is capable of performing its normal function.

`DEPENDENCY`

`public static final com.sun.ctmgx.moh.AvailabilityStatus` **`DEPENDENCY`**

Indicates the object is in dependency state.

`FAILED`

`public static final com.sun.ctmgx.moh.AvailabilityStatus` **`FAILED`**

Indicates the object is in failed state.

`INTEST`

`public static final com.sun.ctmgx.moh.AvailabilityStatus` **`INTEST`**

Indicates the object is under test.

`NOTINSTALLED`

`public static final com.sun.ctmgx.moh.AvailabilityStatus` **`NOTINSTALLED`**

Indicates the object is not properly configured

`OFFLINE`

public static final com.sun.ctmgx.moh.AvailabilityStatus **OFFLINE**

Indicates the object is in offline state.

`POWEROFF`

public static final com.sun.ctmgx.moh.AvailabilityStatus **POWEROFF**

Indicates the object is in power off state.

`UNKNOWN`

public static final com.sun.ctmgx.moh.AvailabilityStatus **UNKNOWN**

Indicates the object is unknown.

Methods

`equals(Object)`

public Boolean **equals**(java.lang.Object obj)

Overrides:

`equals` in class `Object`

`hashCode()`

public int **hashCode**()

Overrides:

`hashCode` in class `Object`

`intValue()`

public int **intValue**()

`toString()`

public java.lang.String **toString**()

Overrides:

`toString` in class `Object`

EquipmentHolderType

Declaration

`public class` **`EquipmentHolderType`** extends java.lang.Object implements `java.io.Serializable`

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.EquipmentHolderType
```

All Implemented Interfaces:

`java.io.Serializable`

Member Summary

Fields

`static EquipmentHolderType`	`DRAWER` This type is currently not supported in Netra CT 410/810.
`static EquipmentHolderType`	`RACK` This type is currently not supported in Netra CT 410/810.
`static EquipmentHolderType`	`SHELF` This type is currently not supported in Netra CT 410/810.
`static EquipmentHolderType`	`SLOT` This type is currently not supported in Netra CT 410/810.

Methods

`Boolean`	`equals(java.lang.Object obj)`
`int`	`hashCode()`
`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

DRAWER

`public static final com.sun.ctmgx.moh.EquipmentHolderType` **`DRAWER`**

This type is currently not supported in Netra CT 410/810.

RACK

`public static final com.sun.ctmgx.moh.EquipmentHolderType` **`RACK`**

This type is currently not supported in Netra CT 410/810.

SHELF

`public static final com.sun.ctmgx.moh.EquipmentHolderType` **`SHELF`**

This type is currently not supported in Netra CT 410/810.

SLOT

`public static final com.sun.ctmgx.moh.EquipmentHolderType` **`SLOT`**

Holder type SLOT such as disk slot, fan slot, power supply slot, or CPCI slots.

Methods

equals(Object)

`public Boolean` **`equals`**`(java.lang.Object obj)`

Overrides:

`equals` in class `Object`

hashCode()

`public int` **`hashCode`**`()`

Overrides:

`hashCode` in class `Object`

intValue()

`public int` **`intValue`**`()`

toString()

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `Object`

LogFullAction

Declaration

`public class` **`LogFullAction`** extends `java.lang.Object` implements `java.io.Serializable`

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.LogFullAction
```

All Implemented Interfaces

`java.io.Serializable`

Description

This class describes the action to perform when the log is full. For Netra CT 410/CT 810, this class is not supported.

Member Summary

Fields

`static LogFullAction`	`HALT`
`static LogFullAction`	`WRAP`

Methods

`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

`HALT`

`public static final LogFullAction` **`HALT`**

`WRAP`

`public static final LogFullAction` **`WRAP`**

Methods

`intValue()`

```
public int intValue()
```

`toString()`

```
public java.lang.String toString()
```

Overrides: `toString` **in class** `java.lang.Object`

MohNames

Declaration

```
public class MohNames extends java.lang.Object
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.MohNames
```

Description

This class defines the public constants and static variables for MOH user to communicate to the MBean server. For example:

- A MOH user might need to know the object name of the ContainmentTreeMBean to traverse the physical topology.
- A MOH user registers as a listener on EFD using the EFD's object name kept in a public static variable.
- A MOH user might need to know the object name of the Software Monitor Service to discover all the software services in the system.

Member Summary

Fields	
`static java.lang.String`	`CLASS_NAME_ASP` AlarmSeverityProfile Class Name.
`static java.lang.String`	`CLASS_NAME_EFD` EFD Class Name.
`static java.lang.String`	`CLASS_NAME_SOFTWAREMONITOR` SoftwareMonitor Class Name.

Member Summary *(Continued)*	
`static java.lang.String`	`DESCR_ALARM` Notification Description: AlarmNotification
`static java.lang.String`	`DESCR_ATTRIBUTE_CHANGE` Notification Description: AttributeChangeNotification
`static java.lang.String`	`DESCR_OBJECT_CREATION` Notification Description: ObjectCreationNotification
`static java.lang.String`	`DESCR_OBJECT_DELETION` Notification Description: ObjectDeletionNotification
`static java.lang.String`	`DESCR_STATE_CHANGE` Notification Description: StateChangeNotification
`static javax.management.ObjectName`	`MOH_CONTAINMENT_TREE` The object name of the only instance of ContainmentTreeMBean
`static javax.management.ObjectName`	`MOH_DEFAULT_ASP` The object name used by the MOH implementation for registering the default Alarm Severity Profile MBean.
`static javax.management.ObjectName`	`MOH_DEFAULT_EFD` The object name used by the MOH implementation for registering the default EFD MBean.
`static javax.management.ObjectName`	`MOH_SOFTWARE_MONITOR` The object name used by the MOH implementation for registering the Software Monitor MBean.

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

CLASS_NAME_ASP

`public static final java.lang.String` **`CLASS_NAME_ASP`**

AlarmSeverityProfile Class Name.

CLASS_NAME_EFD

`public static final java.lang.String` **`CLASS_NAME_EFD`**

EFD Class Name.

CLASS_NAME_SOFTWAREMONITOR

`public static final java.lang.String` **`CLASS_NAME_SOFTWAREMONITOR`**

SoftwareMonitor Class Name.

DESCR_ALARM

`public static final java.lang.String` **`DESCR_ALARM`**

Notification Description: AlarmNotification

DESCR_ATTRIBUTE_CHANGE

`public static final java.lang.String` **`DESCR_ATTRIBUTE_CHANGE`**

Notification Description: AttributeChangeNotification

DESCR_OBJECT_CREATION

`public static final java.lang.String` **`DESCR_OBJECT_CREATION`**

Notification Description: ObjectCreationNotification

DESCR_OBJECT_DELETION

`public static final java.lang.String` **`DESCR_OBJECT_DELETION`**

Notification Description: ObjectDeletionNotification

DESCR_STATE_CHANGE

`public static final java.lang.String` **`DESCR_STATE_CHANGE`**

Notification Description: StateChangeNotification

MOH_CONTAINMENT_TREE

`public static javax.management.ObjectName` **`MOH_CONTAINMENT_TREE`**

The object name of the only instance of ContainmentTreeMBean. The value is `DOMAIN + ":name=ContainmentTree"`

MOH_DEFAULT_ASP

`public static javax.management.ObjectName` **`MOH_DEFAULT_ASP`**

The object name used by the MOH implementation for registering the default Alarm Severity Profile MBean. The value is `DOMAIN + ":name= AlarmSeverityProfile,id=0"`.

`MOH_DEFAULT_EFD`

`public static javax.management.ObjectName` **`MOH_DEFAULT_EFD`**

The object name used by the MOH implementation for registering the default EFD MBean.

The value is `DOMAIN + ":name=EFD"`.

`MOH_SOFTWARE_MONITOR`

`public static ObjectName` **`MOH_SOFTWARE_MONITOR`**

The object name used by the MOH implementation for registering the Software Monitor MBean. The value is `DOMAIN + ":name=softwaremonitor"`.

ObjectCreationNotification

Declaration

`public class` **`ObjectCreationNotification`** extends `TopologyChangeNotification`

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--javax.management.Notification
              |
              +--com.sun.ctmgx.moh.TopologyChangeNotification
                    |
                    +--com.sun.ctmgx.moh.ObjectCreationNotification
```

Description

This class defines the creation notifications sent by MBeans.

It is up to the parent MBean, which creates the child MBean, to send the object creation notification, so the `NotificationBroadcaster` interface has to be implemented by the parent MBean.

Member Summary

Fields

`static java.lang.String`	`OBJECT_CREATION` Notification type denoting that an MBean has been created.

Constructors

	`ObjectCreationNotification(java.lang.Object source, long sequenceNumber, java.lang.String message, javax.management.ObjectName child, javax.management.ObjectName parent)` Creates an ObjectCreationNotification object specifying the object names of the created MBean and the parent of the created MBean in the containment hierarchy.

Methods

`javax.management.ObjectName`	`getChildMBeanName()` Gets the object name of the created or deleted MBean.
`javax.management.ObjectName`	`getParentMBeanName()` Gets the object name of the parent of the created or deleted MBean in the containment hierarchy.

Inherited Member Summary

Methods inherited from class `javax.management.Notification`

`getMessage, getSequenceNumber, getSource, getTimeStamp, getType, getUserData, setSequenceNumber, setSource, setTimeStamp, setUserData`

Methods inherited from class `java.lang.Object`

`toString`

Methods inherited from class `java.lang.Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

OBJECT_CREATION

```
public static final java.lang.String OBJECT_CREATION
```

Notification type denoting that an MBean has been created.

The value of this type string is `netract.moh.object.creation`.

Constructors

`ObjectCreationNotification`

public **ObjectCreationNotification**(java.lang.Object *source*, long *sequenceNumber*, java.lang.String *message*, javax.management.ObjectName *child*, javax.management.ObjectName *parent*)

Creates an ObjectCreationNotification object specifying the object names of the created MBean and the parent of the created MBean in the containment hierarchy.

Parameters:

source – The notification source, that is, the MBean that emits the notification.

sequenceNumber – The notification sequence number within the source object.

message – A String containing the message of the notification.

child – The object name of the created MBean, that is, the child MBean in the containment hierarchy.

parent – The object name of the parent MBean in the containment hierarchy.

Methods

`getChildMBeanName()`

public javax.management.ObjectName **getChildMBeanName()**

Gets the object name of the created or deleted MBean.

Returns:

The `ObjectName` of the created or deleted MBean.

`getParentMBeanName()`

public javax.management.ObjectName **getParentMBeanName()**

Gets the object name of the parent of the created/deleted MBean in the containment hierarchy.

Returns:

The `ObjectName` of the parent MBean.

ObjectDeletionNotification

Declaration

`public class ObjectDeletionNotification` extends **`TopologyChangeNotification`**

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--javax.management.Notification
              |
              +--com.sun.ctmgx.moh.TopologyChangeNotification
                    |
                    +-com.sun.ctmgx.moh.ObjectDeletionNotification
```

Description

This class defines the deletion notifications sent by MBeans.

Member Summary

Fields	
`static java.lang.String`	`OBJECT_DELETION` Notification type denoting that an MBean has been deleted.
Constructors	
	`ObjectDeletionNotification(java.lang.Object source, long sequenceNumber, java.lang.String message, javax.management.ObjectName child, javax.management.ObjectName parent)` Creates an ObjectDeletionNotification object specifying the object names of the deleted MBean and the parent of the deleted MBean in the containment hierarchy.
Methods	
`javax.management.ObjectName`	`getChildMBeanName()` Gets the object name of the created/deleted MBean.
`javax.management.ObjectName`	`getParentMBeanName()` Gets the object name of the parent of the created/deleted MBean in the containment hierarchy.

Inherited Member Summary

Methods inherited from class `javax.management.Notification`

`getMessage, getSequenceNumber, getSource, getTimeStamp, getType, getUserData, setSequenceNumber, setSource, setTimeStamp, setUserData`

Methods inherited from class `java.util.EventObjec`

`toString`

Methods inherited from class `java.lang.Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

OBJECT_DELETION

`public static final java.lang.String` **`OBJECT_DELETION`**

Notification type denoting that an MBean has been deleted.

The value of this type string is `netract.moh.object.deletion`.

Constructors

ObjectDeletionNotification

`public` **`ObjectDeletionNotification`**`(java.lang.Object` *source*`, long` *sequenceNumber*`, java.lang.String` *message*`,` javax.management.`ObjectName` *child*`,` javax.management.`ObjectName` *parent*`)`

Creates an ObjectDeletionNotification object specifying the object names of the deleted MBean and the parent of the deleted MBean in the containment hierarchy.

Parameters:

source – The notification source, that is, the MBean that emits the notification.

sequenceNumber – The notification sequence number within the source object.

message – A String containing the message of the notification.

child – The object name of the deleted MBean, that is, the child MBean in the containment hierarchy.

parent – The object name of the parent MBean in the containment hierarchy.

Methods

`getChildMBeanName()`

`public javax.management.ObjectName **getChildMBeanName**()`

Gets the object name of the created or deleted MBean.

Returns:

The `ObjectName` of the created or deleted MBean.

`getParentMBeanName()`

`public javax.management.ObjectName **getParentMBeanName**()`

Gets the object name of the parent of the created or deleted MBean in the containment hierarchy.

Returns:

The `ObjectName` of the parent MBean.

OperationalState

Declaration

```
public class OperationalState extends java.lang.Object
implements java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.OperationalState
```

All Implemented Interfaces

`java.io.Serializable`

Description

This class defines the operational states of a device (equipment or plug-in):

- `ENABLED`: In-service, that is, the device is capable of performing its normal function.
- `DISABLED`: Out-of-service, that is, the device is incapable of performing its normal function.
- `UNKNOWN`: The system is unable to determine the operation state of this device.

Member Summary

Fields

`static OperationalState`	`DISABLED` Device is incapable of performing its normal function.
`static OperationalState`	`ENABLED` Device is capable of performing its normal function.
`static OperationalState`	`UNKNOWN` State of this device can not be determined.

Methods

`Boolean`	`equals(java.lang.Object obj)`
`int`	`hashCode()`
`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

`DISABLED`

`public static final com.sun.ctmgx.moh.OperationalState` **`DISABLED`**

Device is incapable of performing its normal function.

`ENABLED`

`public static final com.sun.ctmgx.moh.OperationalState` **`ENABLED`**

Device is capable of performing its normal function.

`UNKNOWN`

`public static final com.sun.ctmgx.moh.OperationalState` **`UNKNOWN`**

State of this device cannot be determined.

Methods

`equals(Object)`

`public Boolean equals(java.lang.Object obj)`

Overrides:

`equals` in class `Object`

`hashCode()`

`public int hashCode()`

Overrides:

`hashCode` in class `Object`

`intValue()`

`public int intValue()`

`toString()`

`public java.lang.String toString()`

Overrides: `toString` **in class** `Object`

SlotStatus

Declaration

```
public class SlotStatus extends java.lang.Objec implements
java.io.Serializable
```

```
java.lang.Object
  |
  +--com.sun.ctmgx.moh.SlotStatus
```

All Implemented Interfaces

`java.io.Serializable`

Description

This class defines the status of the slot object.

Member Summary

Fields

`static SlotStatus`	`EMPTY` No card or device is plugged into this slot.
`static SlotStatus`	`FULL` A card or device is plugged into this slot.

Methods

`Boolean`	`equals(java.lang.Object obj)`
`int`	`hashCode()`
`int`	`intValue()`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `Object`

`getClass(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

EMPTY

`public static final com.sun.ctmgx.moh.SlotStatus` **`EMPTY`**

No card or device is plugged into this slot.

FULL

`public static final com.sun.ctmgx.moh.SlotStatus` **`FULL`**

There is a card or device plugged into this slot.

Methods

equals(Object)

`public Boolean` **`equals`**`(java.lang.Object obj)`

Overrides:

`equals` in class `Object`

`hashCode()`

`public int` **`hashCode`**`()`

Overrides:

`hashCode` in class `Object`

`intValue()`

`public int` **`intValue`**`()`

`toString()`

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `Object`

StateChangeNotification

Declaration

```
public class StateChangeNotification extends
javax.management.AttributeChangeNotification
```

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--javax.management.Notification
              |
              +--javax.management.AttributeChangeNotification
                    |
                     +--com.sun.ctmgx.moh.StateChangeNotification
```

Description

This class defines the state change notifications sent by MBeans.

It is up to the MBean owning the state attribute of interest to create and send state change notifications when the state attribute change occurs, so the `NotificationBroadcaster` interface has to be implemented by any MBean interested in sending state change notifications.

Member Summary

Fields

`static java.lang.String`	`STATE_CHANGE` Notification type that indicates that the observed MBean state attribute value has changed.

Constructors

	`StateChangeNotification(java.lang.Object source, long sequenceNumber, java.lang.String message, java.lang.String attributeName, java.lang.String attributeType, java.lang.Object oldValue, java.lang.Object newValue)` Constructs a state change notification object whose type string is netract.moh.state.change In addition to the information common to all notifications, the caller must supply the name and type of the attribute, as well as its old and new values.

Methods

`java.lang.String`	`getType()` Gets notification type. The method of `javax.management.Notification` is overridden to return STATE_CHANGE.

Inherited Member Summary

Methods inherited from class `javax.management.AttributeChangeNotification`

`getAttributeName, getAttributeType, getNewValue, getOldValue`

Methods inherited from class `javax.management.Notification`

`getMessage, getSequenceNumber, getSource, getTimeStamp, getUserData, setSequenceNumber, setSource, setTimeStamp, setUserData`

Methods inherited from class `java.util.Event.Object`

`toString`

Methods inherited from class `java.lang.Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`

Fields

STATE_CHANGE

`public static final java.lang.String` **`STATE_CHANGE`**

Notification type which indicates that the observed MBean state attribute value has changed.

The value of this type string is `netract.moh.state.change`.

Constructors

`StateChangeNotification`

`public` **`StateChangeNotification`**`(java.lang.Object` *source*`,`
`long` *sequenceNumber*`, java.lang.String` *message*`,`
`java.lang.String` *attributeName*`, java.lang.String` *attributeType*`,`
`java.lang.Object` *oldValue*`, java.lang.Object` *newValue*`)`

Constructs a state change notification object whose type string is `netract.moh.state.change`. In addition to the information common to all notifications, the caller must supply the name and type of the attribute, as well as the attribute's old and new values.

Parameters:

source – The notification source, that is, the MBean that emits the notification.

sequenceNumber – The notification sequence number within the source object.

message – A String containing the message of the notification.

attributeName – A String specifying the name of the state attribute.

attributeType – A String specifying the type of the state attribute.

oldValue – An object representing the value of the state attribute before the change.

newValue – An object representing the value of the state attribute after the change.

Methods

`getType()`

`public java.lang.String` **`getType`**`()`

Gets notification type. The method of `javax.management.Notification` is overridden to return STATE_CHANGE.

Overrides:

getType in class javax.management.Notification

StateChangeNotificationFilter

Declaration

```
public class StateChangeNotificationFilter extends
javax.management.AttributeChangeNotificationFilter
```

```
java.lang.Object
  |
  +--javax.management.AttributeChangeNotificationFilter
        |
        +--com.sun.ctmgx.moh.StateChangeNotificationFilter
```

All Implemented Interfaces

`javax.management.NotificationFilter, java.io.Serializable`

Description

This class describes the filtering performed on the name of the observed attribute.

It manages a list of enabled attributes for which notifications should be sent when their attribute changes. A method of this class allows users to enable or disable (allow or disallow notifications to be sent) as many attribute names as required.

Member Summary	
Constructors	
	`StateChangeNotificationFilter()`
Methods	
`Boolean`	`isNotificationEnabled(javax.management.Notification notification)` This method is invoked before sending the specified notification to the listener.

Inherited Member Summary

Methods inherited from class `javax.management.AttributeChangeNotificationFilter`

`disableAllAttributes, disableAttribute, enableAttribute, getEnabledAttributes`

Methods inherited from class `java.lang.Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

`StateChangeNotificationFilter`

`public` **`StateChangeNotificationFilter`**`()`

Methods

`isNotificationEnabled(Notification)`

`public Boolean`
`isNotificationEnabled``(javax.management.Notification` *notification*`)`

This method is invoked before sending the specified notification to the listener.

This filter compares the state attribute name of the specified state change notification with each enabled attribute name. If the attribute name equals one of the enabled attribute names, the notification must be sent to the listener so this method returns `true`.

Overrides:

`isNotificationEnabled` in class `javax.management.AttributeChangeNotificationFilter`

Parameters:

notification – The attribute change notification to be sent.

Returns:

`true` if the notification has to be sent to the listener, `false` otherwise.

CHAPTER 5

Simple Network Management Protocol

This chapter describes the Netra CT server Simple Network Management Protocol (SNMP) support, and provides a useful example. This chapter contains the following sections:

SNMP Overview

The most widespread legacy architecture for network and device management is SNMP, for which the Java DMK provides a complete toolkit. This gives you the advantages of developing both Java Dynamic Management agents and managers that are interoperable with existing management systems.

SNMP network protocol enables devices to be managed remotely by a Network Management Station (NMS). To be managed, a device must have an SNMP agent associated with it. The agent receives requests for data representing the state of the device and provides an appropriate response. The agent can also control the state of the device. Additionally, the agent can generate SNMP traps, which are unsolicited messages sent to selected NMS(s) to signal significant events relating to the device.

The Sun Netra SNMP Management Agent is an intelligent SNMP v2 agent for continuously monitoring key hardware variables. You can generate and collect value-add reports collected by remote monitoring. Using Sun Netra SNMP Management Agent's generic management interface and comprehensive event mechanisms, you can dynamically build configuration and health status data, thus reducing development costs.

Management Information Base (MIB)

To manage and monitor devices, the characteristics of the devices must be represented using a format known to both the agent and the NMS. 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 routing tables, and an example of the latter is a timestamp indicating the time at which the agent was started.

A MIB is a text file, written in abstract syntax notation one (ASN.1) notation, which describes the variables containing the information that SNMP can access. The variables described in a MIB, which are also called MIB objects, are the items that can be monitored using SNMP. There is one MIB object for each element being monitored. All MIBs are, in fact, part of one large hierarchical structure, with leaf nodes containing unique identifiers, data types, and access rights for each variable and the paths providing classifications. A standard path structure includes branches for private subtrees.

For reference, the structure of the MIBs for SNMPv2 is defined by its Structure of Management Information (SMI) defined in the RFC2578 document. This SMI defines the syntax and basic data types available to MIBs. The Textual Conventions (type definitions) defined in the RFC2579 document define additional data types and enumerations.

Before an NMS can manage a device through its agent, the MIB corresponding to the data presented by the agent must be loaded into the NMS. The mechanism for doing this varies depending on the implementation of the network management software. This gives the NMS the information required to address and correctly interpret the data model presented by the agent. Note that MIBs can reference definitions in other MIBs, so to use a given MIB, it might be necessary to load others.

Object Identifiers (OIDs)

The MIB defines a virtual datastore accessible by way of the SNMP software, the content being provided either by corresponding data maintained by the agent, or by the agent obtaining the required data on demand from the managed device. For writes of data by the NMS to this virtual data, the agent typically performs some action affecting the state either of itself or the managed device.

To address the content of this virtual datastore, the MIB is defined in terms of object identifiers (OIDs) which uniquely identify each data entry. An OID consists of an hierarchically arranged sequence of integers providing a unique name space. Each assigned integer has a associated text name. For example, the OID `1.3.6.1` corresponds to the OID `iso.org.dod.internet` and `1.3.6.1.4` corresponds to the OID `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 `netraCt`, to all object names defined therein.

All addressable objects defined in the MIB have associated maximum access rights (for instance, read-only or read-write), which determine what operations the NMS permits the operator to attempt. The agent can limit access rights as required; that is, it is able to refuse writes to objects that are considered read-write. This refusal can be done on the grounds of applicability of the operation to the object being addressed, or on the basis of security restrictions that can limit certain operations to restricted sets of NMS. The mechanism used to communicate security access rights is *community strings*. These text strings, such as `private` and `public`, are passed with each SNMP data request.

Much of the data content defined by MIBs is of a tabular form, organized as entries consisting of a sequence of objects (each with their 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`), or a more complex, multidimensional, instance specifier such as an IP address and port number (for example, `127.0.0.1, 1234`). In either case, a specific data item within a table is addressed by specifying the OID giving its prefix (for example, `myFanTable.myFanEntry.myCurrentFanSpeed`) with a suffix instance specifier (for example, `127.0.0.1.1234` from the previous example) to give `myFanTable.myFanEntry.myCurrentFanSpeed.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. The SMI defining the MIB syntax provides an important capability whereby tables can be extended to add additional entries, effectively adding extra columns to the table. This is achieved by defining a table with an INDEX clause that is a duplicate of that of the table being extended.

Netra CT System SNMP Representation

The Netra CT software uses these SNMP MIBs to present the network information model:

- ENTITY-MIB (RFC 2037)
- IF-MIB (RFC 2863)
- SUN-SNMP-NETRA-CT-MIB
- HOST-RESOURCES-MIB (RFC 2790)

ENTITY-MIB

The ENTITY-MIB is defined by the IETF standard RFC2037. The ENTITY-MIB provides a mechanism for presenting hierarchies of physical entities using SNMP tables.

The Netra CT information model uses the ENTITY-MIB to provide:

- A hierarchy of hardware resources—relationships between managed objects
- Common hardware resource characteristics—a mapping of common attributes from the GNIM Top, Equipment, and Termination Point classes

This information is presented using SNMP tables:

- Physical Entity Table (*entPhysicalTable*)

  This table contains one row per hardware resource. These rows are called e*ntries,* and a particular row is referred to as an *instance*. Each entry contains the physical class (*entPhysicalClass*) and common characteristics of the hardware resource. Each entry has a unique index (*entPhysicalIndex*) and contains a reference (*entPhysicalContainedIn*) that points to the row of the hardware resource which acts as the *container* for this resource.

FIGURE 5-1 and TABLE 5-1 show how an example hierarchy of hardware resources are presented using the ENTITY-MIB.

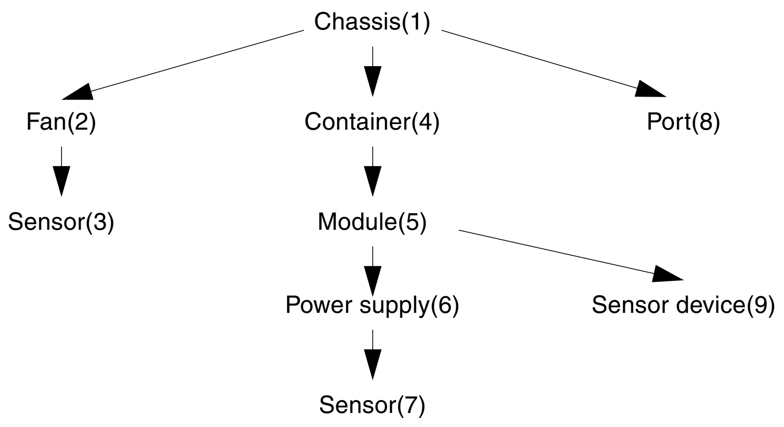


FIGURE 5-1 Hardware Resource Hierarchy

TABLE 5-1 Physical Entity Table

entPhysicalIndex	entPhysicalClass	entPhysicalContainedIn	. . .
1	chassis	0	. . .
2	fan	1	. . .
3	sensor	2	. . .
4	container	1	. . .
5	module	4	. . .
6	power supply	5	. . .
7	sensor	6	. . .
8	port	1	. . .
9	other	5	. . .
10	other	5	. . .

The Netra CT Management Agent uses values for *entPhysicalIndex* and *ifIndex* that might not be contiguous, but are within the range of permitted values.

IF-MIB

The IF-MIB is defined by the IETF standard RFC 2863. The IF-MIB provides information about the network interfaces of the server. The information is presented using the ifTable. The ifTable contains a row for each network interface. The ifTable includes columns which describe the interface (ifDescr), indicate the type of interface (ifType), and the indicate the status of the interface (ifOperStatus).

SUN-SNMP-NETRA-CT-MIB

This section describes the `SUN-SNMP-NETRA-CT-MIB`, which is the SNMP version of the Netra CT network element view.

To summarize, the MIB module consists of the following groups:

- "Netra CT Network Element High-Level Objects" on page 184
- "Physical Path Termination Point Table" on page 185
- "Equipment Table" on page 185
- "Plug-in Unit Table" on page 187
- "Hardware Unit to Running Software Relationship Table" on page 188
- "Hardware Unit to Installed Software Relationship Table" on page 188

- "Alarm Severity Identifier Textual Convention" on page 189
- "Alarm Severity Profile Table" on page 190
- "Alarm Severity Table" on page 190
- "Trap Forwarding Table" on page 191
- "Trap Agent MIB Log Table" on page 192
- "Logged Trap Table" on page 192
- "Logged Alarm Table" on page 193
- "MIB Notification Types" on page 194
- "MIB Notifications" on page 195
- "State Change Notification Traps" on page 195
- "Object Creation and Deletion Notification Traps" on page 195
- "Configuration Change Notification Traps" on page 196

A brief descriptions of these modules are provided in the following subsections. sections. For more information, refer to the MIB file which is available as part of the software package at the default location:

```
/opt/SUNWnetract/mgmt2.0/mibs/SUN-SNMP-NETRA-CT-MIB.mib
```

Netra CT Network Element High-Level Objects

The SUN-SNMP-NETRA-CT-MIB module representation of high-level objects in the Netra CT network element (NE) is composed of the elements in TABLE 5-2:

TABLE 5-2 SUN-SNMP-NETRA-CT-MIB Netra CT NE High-Level Objects

Field	**Description**
Vendor	The vendor of the Netra CT network element.
Version	The version of the Netra CT network element.
Start Time	The time at which the agent was last started; in other words, the time at which sysUpTime was zero (0).
Alarm Severity Index	An index into the alarm severity profile table, specifying the severity assignments for Netra CT alarms reported for the Netra CT network element. The default value for this object is zero (0).
Suppress Zero Stats	When the value of this object is true, no entry will be created in any of the historical statistics tables for intervals in which all counts are zero. The default value for this object is true (1).

Physical Path Termination Point Table

The Netra CT Physical Path Termination Point Table extends the entPhysicalTable. Each entry of this table represents a Physical Path Termination Point within the Netra CT NE. The SUN-SNMP-NETRA-CT-MIB module representation of a physical path termination point is composed of the elements shown in TABLE 5-3:

TABLE 5-3 SUN-SNMP-NETRA-CT-MIB Physical Path Termination Point Table

Field	Description
Physical Path Termination Point Hardware Unit Index	Specifies the index of the entry in the entPhysicalTable that represents the device (that is, a card) on which the physical path terminates.
Physical Path Termination Point Port ID	Identifies the port (within the card identified by the hardware unit index) on which the physical path terminates.
Physical Path Termination Point Port Label	Provides the external label string for the physical path TP entry. If there is no label, the value is a zero-length display string.
Physical Path Termination Point Port Alarm Severity Index	Specifies the index of the entry in the communications alarm severity profile table that should be used. The default value of this object is zero (0).

Equipment Table

The Netra CT Equipment Table extends the entPhysicalTable. Each entry in this table represents a piece of equipment within the Netra CT NE that neither is nor accepts a replaceable plug-in unit. The SUN-SNMP-NETRA-CT-MIB module representation of an equipment is composed of the elements shown in TABLE 5-4:

TABLE 5-4 SUN-SNMP-NETRA-CT-MIB Equipment Table

Field	Description
Equipment Administration Status	Used by the administrator to lock and unlock the object.
Equipment Location	The specific or general location of the component.
Equipment Operating Status	Identifies whether or not the component is capable of performing its normal functions.
Equipment Vendor	The vendor of the component.

TABLE 5-4 SUN-SNMP-NETRA-CT-MIB Equipment Table

Field	Description
Equipment Version	The version of the component.
Equipment User Label	A user-friendly name for the piece of equipment. The default value of this object is the null string.
Equipment Alarm Severity Index	An index into the alarm severity profile table, specifying the severity assignments for Netra CT alarms reported for this component. The default value of this object is zero (0).

Equipment Holder Table

The Netra CT Equipment Holder table extends the entPhysicalTable. Each entry in this table represents a component within the Netra CT NE that accepts a replaceable plug-in unit. The SUN-SNMP-NETRA-CT-MIB module representation of an equipment holder is composed of the elements shown in TABLE 5-5:

TABLE 5-5 SUN-SNMP-NETRA-CT-MIB Equipment Holder

Field	Description
Equipment Holder Type	The type of the component.
Equipment Holder Acceptable Types	The types of plug-in units that can be supported by the slot, separated by newline characters. This attribute is present only when the Equipment Holder represents a slot.
Equipment Holder Slot Status	Identifies whether or not a plug-in unit is present in the slot. This attribute is present only when the Equipment Holder represents a slot.
Equipment Holder Label	Provides the external label string for the holder entry. If there is no label, the value is a zero-length display string.
Equipment Holder Software Load	An index into the installed software table, specifying the software that is to be loaded into the plug-in unit whenever an automatic reload of software is needed. This attribute is present only when the Equipment Holder represents a slot.

Plug-in Unit Table

The Plug-In Unit Table extends the entPhysicalTable. Each entry of this table represents a piece of equipment within the Netra CT NE that is inserted into and removed from an Equipment Holder. The SUN-SNMP-NETRA-CT-MIB module representation of a plug-in unit is composed of the elements shown in TABLE 5-6.

TABLE 5-6 SUN-SNMP-NETRA-CT-MIB Plug-In Unit Table

Field	Description
Plug-In Unit Administration Status	Used by the administrator to lock and unlock the object. Values are: up (1) and down (2).
Plug-In Unit Availability Status	Provides further information regarding the state of the component. Value are: available (1), inTest (2), failed (3), powerOff (4), notInstalled (5), offine (6), dependency (7), and unknown (8).
Plug-In Unit Operative Status	Identifies whether or not the component is capable of performing its normal functions. Values are: up (1), down (2), and unknown (3).
Plug-In Unit Vendor	The vendor of the component.
Plug-In Unit Version	The version of the component.
Plug-In Unit Label	Provides the external label string for the plug-in entry. If there is no label, the value is a zero-length display string.
Plug-In Unit Alarm Severity Index	An index into the alarm severity profile table, specifying the severity assignments for Netra CT alarms reported for this component. The default value of this object is zero (0).

Hardware Unit to Running Software Relationship Table

The Netra CT Hardware Unit to Running Software Relationship Table describes the software that is running on each hardware unit in the Netra CT NE. Each entry of this table identifies an entry in the entPhysicalTable and one in the hrSWInstalledTable.

The SUN-SNMP-NETRA-CT-MIB hardware unit to running software relationship table is composed of the elements shown in TABLE 5-7.

TABLE 5-7 SUN-SNMP-NETRA-CT-MIB Hardware Unit to Running Software Relation Table

Field	Description
Hardware Running Software to Hardware Index	The index, in the entPhysicalTable, of the containing hardware unit in this pair.
Hardware Running Software Index	A unique number within the context of the containing hardware unit.
Hardware Running Software to Software Index	An index into the Netra CT Hardware Unit/Running Software relationship table.

Hardware Unit to Installed Software Relationship Table

The Netra CT Hardware Unit to Install Software Relationship Table describes the software that is installed on each hardware unit in the Netra CT NE. Each entry of this table identifies an entry in the entPhysicalTable and one in the hrSWInstalledTable. The SUN-SNMP-NETRA-CT-MIB hardware unit to installed software relationship table is composed of the elements shown in TABLE 5-8.

TABLE 5-8 SUN-SNMP-NETRA-CT-MIB Hardware Unit to Installed Software Relationship Table

Field	Description
Hardware Installed Software to Hardware Index	The index, in the entPhysicalTable, of the containing physical entity in this pair.
Hardware Installed Software Index	A unique number within the context of the containing hardware unit.

TABLE 5-8 SUN-SNMP-NETRA-CT-MIB Hardware Unit to Installed Software Relationship Table

Field	Description
Hardware Installed Software to Software Index	The index, in the hrSWInstalledTable, of the software product represented by this entry.
Hardware to Software Alarm Severity Index	An index into the alarm severity profile table, specifying the severity assignments for Netra CT alarms reported for this piece of software installed on the hardware unit. The default value of this object is zero.
Hardware Installed Software to Hardware Index	The index, in the entPhysicalTable, of the containing physical entity in this pair.

Alarm Severity Identifier Textual Convention

The SUN-SNMP-NETRA-CT-MIB alarm severity identifier textual conventions consist of the elements shown in TABLE 5-9.

TABLE 5-9 SUN-SNMP-NETRA-CT-MIB Alarm Severity Identifier Textual Conventions

Field	Description
Alarm Log Severity	The value of this object identifies the severity of an alarm in the log. Values are: cleared (-1), indeterminate (0), critical (1), major (2), minor (3), and warning (4).
Alarm Severity	The value of this object identifies the severity of an alarm that has occurred. Values are: indeterminate (0), critical (1), major (2), minor (3), and warning (4). (Note that there is no value corresponding to 'cleared'.)

Alarm Severity Profile Table

The Netra CT alarm severity profile table specifies which profiles exist. Creating or deleting an entry in this table automatically creates or deletes the corresponding entries in the netraCtAlarmSeverityTable. Each entry of this table represents a group of severities, one for each alarm type in the communications alarm group. The SUN-SNMP-NETRA-CT-MIB alarm severity profile table consists of the elements shown in TABLE 5-10.

TABLE 5-10 SUN-SNMP-NETRA-CT-MIB Alarm Severity Profile Table

Field	Description
Alarm Severity Profile Index	A number identifying this alarm severity profile.
Alarm Severity Profile Row Status	This object is used to create a new row or to delete an existing row in the table.

Alarm Severity Table

The Netra CT alarm severity table associates profile index and trap ID pairs with severities to be used for Netra CT alarm traps that have occurred. (Note that this table does not apply to cleared alarms). An entry in this table associates an alarm severity profile index/trap ID pair with a severity. Deleting a particular profile's row in the alarm severity profile table deletes all rows in this table with the same profile index. Conceptually, rows corresponding to all possible trap IDs are created in this table when a new alarm severity profile is created, but the agent returns a default value except for those few traps for which values have been set. The alarm severity table elements are listed in TABLE 5-11.

TABLE 5-11 SUN-SNMP-NETRA-CT-MIB Alarm Severity Table

Field	Description
Alarm Severity Trap ID	The ID of the trap type to which this entry applies.

Trap Forwarding Table

The Netra CT Trap forwarding discriminator table specifies which traps will be sent to which management system. Each entry of this table contents information about a group of traps to be sent to a particular IP address. This is used as the value of the object netraCtForwardedTrapObject when traps from all objects are to be forwarded, or when there is only one object of the type that forwards the specified trap type. The elements for this table are shown in TABLE 5-12.

TABLE 5-12 SUN-SNMP-NETRA-CT-MIB Trap Forwarding Table

Field	Description
Trap Forwarding Index	A number identifying the Trap forwarding discriminator.
Trap Forwarding Destination	The IP address to which traps identified by this table entry should be sent.
Forwarded Trap ID	The ID of the trap type to which this entry applies. The special value {0 0} indicates that this entry applies to all traps.
Forwarded Trap Object	The object to which this entry applies. By convention, this is the name of the first object in the row in the table referenced. The special value {0 0} indicates that traps of this type from all objects of the type that can generate it. It should also be used when traps from the Netra CT NE are to be specified.
Trap Forwarding Port	The UDP port on the specified management system to which traps identified by this entry should be sent.
Lowest Forwarded Severity	The lowest severity of traps of this type from the specified object that should be sent to this address. This object has significance only if the trap type specified has a severity associated with it.
Forwarded Indeterminate	When this object has the value TRUE, traps with indeterminate severity will be forwarded to the specified event. This object has significance only if the trap type specified has a severity associated with it.
Trap Forwarding Row Status	This object is used to create a new row or to delete an existing row in the table.

Trap Agent MIB Log Table

The Netra CT Trap Agent MIB Log Table defines the trap logs currently maintained by the agent. The management system creates entries in this table to specify which types of traps, from which Netra CT network elements, should be logged. Deleting an entry in this table deletes all entries in the corresponding log. Each entry of this table represents information about a single trap log.

The SUN-SNMP-NETRA-CT-MIB trap agent MIB log table consists of the elements shown in TABLE 5-13.

TABLE 5-13 SUN-SNMP-NETRA-CT-MIB Trap Agent MIB Log Table

Field	Description
Trap Log Source	The IP address of the SNMP agent whose traps are stored in this log.
Trap Log Type	The type of traps stored in this log. Values are: objectCreated (1), objectDeleted (2), configChange (3), stateChange (4), and alarm (5)
Trap Log Administrative Status	The management system uses this object to stop and start the operations of this object. Values are: up (1) and down (2). Default is up (1).
Trap Log Operational Status	Indicates whether or not the log is capable of performing its normal operations.
Trap Log Full Action	Indicates the action that should be performed when no more log entries can be created due to a log-full condition. If the value of this object is wrap (2), each new log entry will cause the deletion of the oldest entry still in the log, for as long as the log is still full. Value are: halt (1) and wrap (2). Default is wrap (2).
Trap Log Row Status	This object is used to create a new row or to delete an existing row in the table.

Logged Trap Table

The Netra CT logged trap table contains information about a single trap in the log. Entries in this table are created automatically but can be deleted by the management system. Entries that represent alarm log types are augmented by the netraCtLoggedAlarmEntry table. Each entry in this table is a unique number identifying this entry in the log. When the maximum value for this object has been reached, it wraps around to 0. The SUN-SNMP-NETRA-CT-MIB Logged Trap Table is used to maintain the traps logged and consists of the elements shown in TABLE 5-14.

TABLE 5-14 SUN-SNMP-NETRA-CT-MIB Logged Trap Table

Field	Description
Logged Trap Time	The time at which this trap was logged.
Logged Trap ID	The type of trap to which this entry applies. Together with the logged trap ID object, this object specifies the entity to which this logged trap referred.
Logged Trap Object	The object to which this entry applies. By convention, this is the name of the first object in the row in the table referenced. Together with the logged trap ID object, this object specifies the entity to which this logged trap referred. The special value {0 0} indicates that the trap refers to the Netra CT NE entity itself.
Logged Trap Row Status	This object is used to delete an existing row in the table. Note that the only value to which a management system can set this object is destroy(6).

Logged Alarm Table

The Netra CT Logged Alarm Trap table is used to maintain extra information for logged traps that represent alarm types. Entries in this table contain information about the alarm-specific attributes of a single trap in the log. The SUN-SNMP-NETRA-CT-MIB Logged Alarm Table consists of the elements shown in TABLE 5-15.

TABLE 5-15 SUN-SNMP-NETRA-CT-MIB Logged Alarm Table

Field	Description
Logged Alarm Severity	The perceived severity of the alarm, as specified by the agent that generated it.
Logged Alarm Backed Up	If the value of this object is true, the agent reported in this trap that the failed object had been backed up. This object is only present if it was included in the alarm trap corresponding to this log entry.

TABLE 5-15 SUN-SNMP-NETRA-CT-MIB Logged Alarm Table

Field	Description
Logged Alarm Backed Up Object	Indicates the object that provided back-up services to the failed object. This object is only present if it was included in the alarm trap corresponding to this log entry.
Logged Alarm Specific Problem	Indicates further refinements to the problem identified by the alarm type. If more than one specific problem is described in this object, the problem descriptions are separated by newline characters. This object is only present if it was included in the alarm trap corresponding to this log entry.
Logged Alarm Repair Act	Indicates proposed repair actions reported by the agent for the problem identified by the alarm. If more than one action is described in this object, the problem descriptions are separated by newline characters. This object is only present if it was included in the alarm trap corresponding to this log entry.

MIB Notification Types

MIB notification types consist of auxiliary definitions for alarms. Except for perceived severity, the objects shown in TABLE 5-16 can be optionally appended to any alarm notification.

TABLE 5-16 MIB Notification Types

Field	Description
Trap Alarm Severity	The perceived severity of the alarm, as specified by the agent that generated it.
Trap Alarm Backed Up	If the value of this object is true, the failed object has been backed up.
Trap Alarm Back-Up Object	Indicates the object that provided back-up services to the failed object.
Trap Alarm Specific Problem	Indicates further refinements to the problem identified by the alarm type. If more than one specific problem is described in this object, the problem descriptions are separated by newline characters.
Trap Alarm Repair Act	Indicates proposed repair actions reported by the agent for the problem identified by the alarm. If more than one action is described in this object, the problem descriptions are separated by newline characters.

MIB Notifications

Note that index values for interfaces, hardware units, and other objects can be derived from the instance values of the objects included in the notifications. For example, the ifIndex value for an interface can be derived from the ifOperStatus instance value, and the entPhysicalIndex value can be derived from any of the entPhysicalContainedIn, entPhysicalParentRelPos, and entPhysicalClass instance values.

State Change Notification Traps

The SUN-SNMP-NETRA-CT-MIB state change notification trap table consists of the elements shown in TABLE 5-17.

TABLE 5-17 SUN-SNMP-NETRA-CT-MIB State Change Notification Traps

Field	**Description**
Hardware Unit Up	Indicates that the operational state of the specified hardware unit has transitioned to up.
Hardware Unit Down	Indicates that the operational state of the specified hardware unit has transitioned to down.

Object Creation and Deletion Notification Traps

The SUN-SNMP-NETRA-CT-MIB object creation and deletion notification traps table consists of the elements shown in TABLE 5-18.

TABLE 5-18 SUN-SNMP-NETRA-CT-MIB Object Creation and Deletion Notification Traps

Field	**Description**
Hardware Unit Created	Indicates that the specified hardware unit has been installed at the specified location.
Hardware Unit Deleted	Indicates that the specified hardware unit has been removed or uninstalled from the specified location.
Installed Software Created	Indicates that the specified software package has been installed.

TABLE 5-18 SUN-SNMP-NETRA-CT-MIB Object Creation and Deletion Notification Traps

Field	Description
Installed Software Deleted	Indicates that the specified software package has been removed.
Running Software Created	Indicates that the specified software has been started.
Running Software Deleted	Indicates that the specified software has been stopped.

Configuration Change Notification Traps

The SUN-SNMP-NETRA-CT-MIB configuration change notification traps table consists of the elements shown in TABLE 5-19.

TABLE 5-19 SUN-SNMP-NETRA-CT-MIB Configuration Change Notification Traps

Field	Description
Interface Changed	Indicates that the configuration of the interface has been changed.
Hardware Unit Changed	Indicates that the specified hardware unit configuration has changed
Installed Software Changed	Indicates that the specified software package configuration has changed.

See the MIB module file for a complete description of SNMP traps.

Understanding the MIB Variable Descriptions

TABLE 5-20 defines the MIB elements used in MIB module descriptions in the sections of the MIB file. For detailed information about these elements, refer to the RFC2578 document, which can be downloaded from the `http://www.ietf.org` web site.

Note – Not every MIB element is present for every MIB module.

TABLE 5-20 MIB Variable Syntax

MIB Element	Description
Module name	The name of the MIB module.
Module type	The type of ASN.1 macro used for the module. Macro types are the following: • OBJECT-TYPE – Defines the type of the managed object. • NOTIFICATION-TYPE – Defines the information contained within an unsolicited transmission of management information (for example, a trap or a request).
SYNTAX	Defines the data structure of the module.
MAX-ACCESS	Defines whether the module can read, write, and/or create an instance of the object, or to include its value in a notification. Can be one of the following: • not-accessible – Indicates an auxiliary object (objects that are both specified in the INDEX clause of a conceptual row and also columnar objects of the same conceptual row are termed auxiliary objects). • accessible-for-notify – Indicates an object that is accessible only by way of a notification (for example, an SNMP trap). • read-only – Only able to read an instance of the object. • read-write – Able to read and write, but not create an instance of the object. • read-create – Able to read, write, and create an instance of the object provides the maximum level of access (read-create is a superset of read-write).
STATUS	Indicates whether this module definition is current or historic. All of the modules in the `SUN-SNMP-NETRA-CT-MIB` are current.
DESCRIPTION	Describes the function and use of the module.
INDEX	The INDEX clause defines instance identification information for the columnar objects subordinate to that object. Refer to RFC2578 for more information.
Default Value	Defines the default value (DEFVAL) which might be used at the discretion of an SNMP agent when an object instance is created.

For a complete description, see the MIB module in the default location `/opt/SUNWnetract/mgmt2.0/mibs/SUN-SNMP-NETRA-CT-MIB.mib`, delivered as part of the Netra CT software package.

Changing Midplane FRU-ID

This section shows how to change the *locationName* part of FRU-ID.

The Netra CT midplane stores the *locationName*, which is the geographical location of the system, for example, chassis6. This value is stored in the alarm card flash and can be set by the customer. The *locationName* enables system monitoring applications to report specific details.

This example uses an NET-SNMP application to interact with MOH and set the midplane's location to a particular value.

1. **Determine the index of the midplane object from the entPhysicalTable.**

   At the prompt, type the command:

```
snmpwalk -c public -m SUN-SNMP-NETRA-CT-MIB hostName \
entPhysicalDescr
```

Where:

`-c` *community* specifies the community string.

`-m SUN-SNMP-NETRA-CT-MIB` specifies that the Netra CT MIB should be loaded.

hostName is the development system running MOH.

This process and its results are shown in CODE EXAMPLE 5-1

CODE EXAMPLE 5-1 Index of the Midplane Object

```
$snmpwalk -c public -m SUN-SNMP-NETRA-CT-MIB hostName:9161 entPhysicalDescr
ENTITY-MIB::entPhysicalDescr.2 = STRING: 01ae 5405026 Midplane 0000
ENTITY-MIB::entPhysicalDescr.3 = STRING: scb_slot
ENTITY-MIB::entPhysicalDescr.4 = STRING: fan_slot
ENTITY-MIB::entPhysicalDescr.5 = STRING: fan_slot
ENTITY-MIB::entPhysicalDescr.6 = STRING: ps_slot
ENTITY-MIB::entPhysicalDescr.7 = STRING: crtm_slot
ENTITY-MIB::entPhysicalDescr.8 = STRING: cftm_slot
ENTITY-MIB::entPhysicalDescr.9 = STRING: cpci_slot
ENTITY-MIB::entPhysicalDescr.10 = STRING: cpci_slot
ENTITY-MIB::entPhysicalDescr.11 = STRING: cpci_slot
ENTITY-MIB::entPhysicalDescr.12 = STRING: cpci_slot
ENTITY-MIB::entPhysicalDescr.13 = STRING: cpci_slot
ENTITY-MIB::entPhysicalDescr.14 = STRING: prtm_slot
ENTITY-MIB::entPhysicalDescr.15 = STRING: pdu
```

CODE EXAMPLE 5-1 Index of the Midplane Object *(Continued)*

```
ENTITY-MIB::entPhysicalDescr.25 = STRING: 01ae 5016118 scb 0499
ENTITY-MIB::entPhysicalDescr.26 = STRING: ssp_slot
ENTITY-MIB::entPhysicalDescr.27 = STRING: ssp
ENTITY-MIB::entPhysicalDescr.28 = STRING: 01ae 5404931 fan 0499
ENTITY-MIB::entPhysicalDescr.29 = STRING: 01ae 5404931 fan 0499
ENTITY-MIB::entPhysicalDescr.30 = STRING: 01ae 3001535 ps 0399
ENTITY-MIB::entPhysicalDescr.31 = STRING: cftm
ENTITY-MIB::entPhysicalDescr.32 = STRING: 0000 5016123 0101 0000
ENTITY-MIB::entPhysicalDescr.33 = STRING: RJ45
ENTITY-MIB::entPhysicalDescr.35 = STRING: RJ45
ENTITY-MIB::entPhysicalDescr.37 = STRING: RJ45
ENTITY-MIB::entPhysicalDescr.39 = STRING: RJ45
ENTITY-MIB::entPhysicalDescr.41 = STRING: DB15
```

2. **Set the midplane location to the new value of `chassis6` using the following command:**

```
$snmpset -c public -m SUN-SNMP-NETRA-CT-MIB hostName:9161 \
netraCtEquipLocation.1 = chassis6
```

3. **Show the current value of the midplane's location.**

   At the prompt, type the command:

```
$snmpget -c public -m SUN-SNMP-NETRA-CT-MIB hostName:9161 \
netraCtEquipLocation.1
```

The result displays the identifying string of the location of any Netra CT equipment locations, as shown in CODE EXAMPLE 5-2.

CODE EXAMPLE 5-2 Identifying the Midplane's Current Location

```
$snmpwget -c public -m SUN-SNMP-NETRA-CT-MIB hostName:9161 \
netraCtEquipLocation.1
SUN-SNMP-NETRA-CT-MIB::netraCtEquipLocation.1 = STRING: chassis6
```

Setting High Temperature Alarms

An alarm in SNMP is defined as a trap with a severity associated with it. When a `HIGH_TEMPERATURE` alarm (CPU high temperature) occurs, the user's application will receive the SNMP trap netraCtHwHighTempAlarm, and netraCtIfChanged trap for the ifOperStatus of the interface corresponding to the alarm output port. The user's application also will receive alarm clear traps when the condition of alarms are cleared, and an attribute change trap of the ifOperStatus. Refer to "Software Modules in the SNMP View" on page 207 for more information.

The Netra CT alarm card supports three output alarm interfaces. The alarm pins (alarm0, alarm1, alarm2) are statically mapped into severities of critical, major, minor respectively. When an alarm occurs, the corresponding alarm pin is driven high according to the severity of the alarm.

The following example shows how to set the high temperature alarm from the default to major.

▼ To Set the High Temperature Alarm Severity to Major

1. **Create an entry in the netraCtAlarmSevProfileTable.**

   At the prompt, type the command:

```
$snmpset -c public -m SUN-SNMP-NETRA-CT-MIB hostName\
netraCtAlarmSevProfileRowStatus.1 = 4
```

Where:

`-c` *community* specifies the community string.

`-m SUN-SNMP-NETRA-CT-MIB` specifies that the Netra CT MIB should be loaded.

hostName is the development system running MOH.

This process and its result are shown in CODE EXAMPLE 5-3.

CODE EXAMPLE 5-3 Creating an Entry in the Profile Table

```
$snmpset -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161 \
netraCtAlarmSevProfileRowStatus.1 = 4
SUN-SNMP-NETRA-CT-MIB::netraCtAlarmSevProfileRowStatus.1 = INTEGER: active(1)
```

Creating an entry in the netraCtAlarmSevProfileTable also creates an entry in the netraCtAlarmSevTable. The entry in the latter corresponds to the profile entry and translates the high temperature alarm entry into the row of integers shown in CODE EXAMPLE 5-4.

CODE EXAMPLE 5-4 Automatic Entry Created in Corresponding Alarm Severity Table

```
$snmpwalk -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161\
netraCtAlarmSevTable
SUN-SNMP-NETRA-CT-MIB:\
:netraCtAlarmSeverity.1.15.1.3.6.1.4.1.42.2.65.1.1.1.2.0.34 = INTEGER:\
minor(3)
End of MIB
```

2. **Set the severity of the netraCtHighTempAlarm for this profile.**

   At the prompt, type the command:

```
$ snmpset -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161\
netraCtAlarmSeverity.1.15.1.3.6.1.4.1.42.2.65.1.1.1.2.0.34 = 2
```

   Where:

   `1.3.6.1.4.1.42.2.65.1.1.1.2.0.34` represents the string ‘netraCtHighTempAlarm’

   The entry at = (in this example, `2`) establishes a major alarm severity.

   The result is shown in CODE EXAMPLE 5-5.

CODE EXAMPLE 5-5 Setting the Alarm Severity for the Profile Table

```
$ snmpset -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161\
netraCtAlarmSeverity.1.15.1.3.6.1.4.1.42.2.65.1.1.1.2.0.34 = 2
SUN-SNMP-NETRA-CT-MIB:\
:netraCtAlarmSeverity.1.15.1.3.6.1.4.1.42.2.65.1.1.1.2.0.34 = INTEGER:
major(2)
```

3. **Set netraCtEquipAlarmSeverityIndex of the thermistor entry to correspond with the netraCtAlarmSevProfile entry from the netraCtAlarmSevProfileTable.**

   At the prompt, type the command:

```
$ snmpset -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161 \
netraCtEquipAlarmSeverityIndex.2 = 1
```

This example uses the **netraCtAlarmSevProfileTable** entry from CODE EXAMPLE 5-3. The index of that entry was the integer 1 in the statement: **netraCtAlarmSevProfileRowStatus.1**. The result of this process is shown in CODE EXAMPLE 5-6.

CODE EXAMPLE 5-6 Setting the Index Entry Corresponding to the Thermistor

```
$snmpset -c public -m SUN-SNMP-NETRA-CT-MIB localhost:9161 \
netraCtEquipAlarmSeverityIndex.2 = 1
SUN-SNMP-NETRA-CT-MIB::netraCtEquipAlarmSeverityIndex.2 = INTEGER: 1
```

When the CPU temperature returns to normal, the alarms are cleared automatically. For further information, refer to the `SUN-SNMP-NETRA-CT-MIB`.

CHAPTER 6

Managed Object Hierarchy Software Modules

This chapter provides a high-level description of the Release 2 Management Object Hierarchy (MOH) Software Modules for the Netra CT platform. It describes the software module interfaces and their major internal modules. It consists of:

Software Module Design

The software services in the system are monitored by software modules which are part of the Information Module layer objects.

Software services are either reliable services (such as RNFS, RBS, or CGTP) or unreliable services (such as TFTP, or NIS). The software services can be a software subsystem such as a network stack (TCP, IP, UDP); an I/O driver such as a network driver; or network processes or network daemons such as NFS.

Some software services are only available on certain CPU boards. For example, CGTP is available for both the host and the satellite CPU boards, but RBS or RNFS are only available on the host CPU board.

The software module interacts with the OS platform through Java interfaces to:

- Monitor OS platform software services for software status, such as installed/not installed or configured/not configured
- Monitor software service subsystems and daemons for status, such as running /not running
- Provide traps and notifications for events related to the status of software services

The software module also provides APIs for management applications to configure the monitoring of software services, such as setting error thresholds, setting polling intervals, starting and stopping polling, and setting maximum retry-counts for the recovery of the daemons.

Software Services

The software modules monitor the following software services:

CGTP	RDHCP
Ethernet Interface Statistics	RNFS
NIS	SNDR
PMS	TCP, IP, UDP
Platform Management Service (PICLD on Solaris)	TFTP
RDHCP	UFS
RNFS	

Software Module MBeans

This section describes the Software Module MBeans for each of the software services that the MOH Software Modules monitor. As specified by the Java Management extensions for instrumentation, all attributes and operations are explicitly listed in an MBean interface. This interface must have the same name as the class that implements it, followed by the MBean suffix. Since the interface and its implementation are usually in different files, there are two files which make up a standard MBean.

These MBeans and their public APIs provide the management interface to manage the applications. All the specific MBeans below are extended from the SoftwareServiceMbean. For more specific information, refer to the Java documents for the APIs that are distributed as part of the Netra CT MOH package. See "Viewing the Netra CT Management Agent API Online" on page 51 for details.

SoftwareMonitorMBean

The SoftwareMonitorMBean is an object that clients can use to discover all the software services in the system. The SoftwareMonitorMBean contains the method `getSoftwareServiceList()` which returns the list of software services.

DaemonMBean

This class provides the name of the daemon, the state of the daemon, and the daemon recovery try count.

SoftwareServiceMBean

The SoftwareServiceMBean provides the base class from which other ServiceMBeans are extended. The SoftwareServiceMBean provides the following:

- Name of the services
- Status of the service (up or down)
- Getting and setting polling intervals
- Starting the polling
- Stopping the polling
- Getting and setting the number of excessive error intervals. This number is the threshold that determines if an event is sent to a client. If an error count exceeds this number, an error event is sent. There will be no more error events until the error condition disappears or a clear event is sent. For example, assume that the error threshold is set at 5% error per total transaction and the number of excessive intervals is set at 3. If the error exceeds 5% in more than 3 consecutive polling intervals, a file system error event is sent to the client.
- Getting a list of DaemonMBeans that support the service, if any.

NfsServiceMBean

The NfsServiceMBean enables the client to monitor the NFS services. A client can get and set the maximum error threshold, get and set the threshold for excessive error intervals, and get the list of NFS mount failures.

UfsServiceMbean

The UfsServiceMBean enables the client to monitor the UFS services. A client can get and set the maximum threshold of the file system usage percentage, get and set the threshold for the number of excessive usage intervals, and query the list of file systems exceeding the usage threshold.

TcpServiceMBean

The TcpServiceMBean enables the client to monitor the TCP services. A client can get status and statistics for the TCP network layer, get and set intervals and thresholds for gathering the statistics, start and stop polling, and get a list of daemons supporting the service.

UdpServiceMBean

The UdpServiceMBean enables the client to monitor the UPD services. A client can get status and statistics for the UDP network layer, get and set intervals and threshold for gathering the statistics, start and stop polling, and get a list of daemons supporting the service.

IpServiceMBean

The IpServiceMBean enables the client to monitor the IP services. A client can get status and statistics for the IP network layer, get and set intervals and thresholds for gathering the statistics, start and stop polling, and get a list of daemons supporting the service.

EtherIfStatsMBean

The EtherIfStatsMbean monitors the Ethernet drivers, and monitors the interface for transmitter and receiver error counts. A client can set and get the maximum error threshold, set and get the threshold for number of excessive intervals, and query for the list of Ethernet interfaces in error.

CgtpServiceMBean

The CgtpServiceMBean enables the client to monitor the CGTP services. A client can get status and statistics for the IP network layer, list and get state of associated Ethernet physical interfaces, get and set intervals and thresholds for gathering the statistics, start and stop polling, and get a list of daemons supporting the service.

RnfsServiceMBean

The RnfsServiceMBean enables the client to monitor the RNFS services. A client can get status and statistics for the UDP network layer, get and set intervals and thresholds for gathering the statistics, start and stop polling, and get a list of daemons supporting the service.

Software Modules in the SNMP View

The SNMP view is supported through the Host Resources MIB. The SNMP client can query for the software services using the Host Resource Software Running table, and can query for the software services that are installed in the local system using the Host Resource Software Installed table.

Host Resources MIB

The Host Resources MIB is defined in RFC 2790.

Host Resources Running Software Table (hrSWRunTable)

The Host Resources Running Software Table contains information about the software that is running on the network element (for example, NFS, TFTP, and CGTP). When an application or daemon under the monitor is running, the MOH Software Module adds an entry into the hrSWRunTable and will send to the client the netraCtRunningSwCreated trap. When an application or a daemon stops running, the MOH Software Module sends the netraCtRunningSwChanged trap with `hrSWRunStatus` is invalid. The MOH Software Module only deletes the entry from the hrSWRunTable and sends the netraCtRunningSwDeleted trap when the service is uninstalled from the system.

Host Resources Installed Software Table (hrSWInstalledTable)

The Host Resources Installed Software Table contains information about the software installed on the network element (for example, installation packages related to NFS, CGTP, and so on). netraCtInstalledSwCreated, netraCtInstalledSwDeleted and netraCtInstalledSwChanged are traps sent to the client corresponding to the software package installed event, software package uninstalled event, and different version of the existing software package installed event.

SNMP Traps

The SNMP management software has the ability to send traps, or messages, to an application when one or more conditions have been met. Generally, a trap is an unsolicited network packet sent from an agent that usually reports some unexpected error condition.

TABLE 6-1 describes the SNMP traps found in the Netra CT SNMP MIB.

TABLE 6-1 SUN-SNMP-NETRA-CT-MIB Traps

SNMP Trap	Description
netraCtHwHighTempAlarm	Indicates that a high temperature condition has occurred on the hardware unit associated with the specified index.
netraCtHwUnitUp	Indicates that the operational state of the specified hardware unit has transitioned to up.
netraCtHwUnitDown	Indicates that the operational state of the specified hardware unit has transitioned to down.
netraCtHwUnitCreated	Indicates that the specified hardware unit has been installed at the specified location.
netraCtHwUnitDeleted	Indicates that the specified hardware unit has been removed or uninstalled from the specified location.
netraCtInstalledSwCreated	Indicates that the specified software package has been installed.

TABLE 6-1 SUN-SNMP-NETRA-CT-MIB Traps *(Continued)*

SNMP Trap	Description
netraCtInstalledSwDeleted	Indicates that the specified software package has been removed.
netraCtRunningSwCreated	Indicates that the specified software has been started.
netraCtRunningSwDeleted	Indicates that the specified software has been stopped.

TABLE 6-2 defines the standard SNMP traps found in the RFC123-MIB.

TABLE 6-2 RFC1213-MIB Traps

SNMP Trap	Description
coldStart	Signifies that the entity, acting in an agent role, is reinitializing itself and that its configuration might have been altered.
warmStart	Signifies that the entity, acting in an agent role, is reinitializing itself such that its configuration is unaltered.
linkUp	Signifies that the entity, acting in an agent role, has detected that the ifOperStatus object for one of its communication links left the down state and transitioned into some other state (but not into the notPresent state). This other state is indicated by the included value of ifOperStatus.
linkDown	Signifies that the entity, acting in an agent role, has detected that the ifOperStatus object for one of its communication links is about to enter the down state from some other state (but not from the notPresent state). This other state is indicated by the included value of ifOperStatus.

CHAPTER 7

Processor Management Services

This chapter describes the processor management services (PMS) application programming interface (API). This chapter contains the following sections:

PMS Software Overview

The processor management services (PMS) software is an extension to the Netra CT platform services software that addresses the requirements of high-availability (HA) application frameworks. The PMS software enables client applications to manage the operation of the processor nodes within a single Netra CT system or within a cluster of multiple Netra CT systems. A processor node is a combination of CPU blade hardware, CPU memory, I/O interfaces, the operating system that runs on them, and select applications. A PMS cluster can include the alarm card and all of the CPU cards in a single Netra CT system, or it can include a defined group of alarm cards and CPU cards located in multiple systems.

The PMS software provides distributed CPU board resource management infrastructure for clusters of CPU boards. This infrastructure includes low-level administrative control and monitoring, high-level configuration, fault recovery, and user-interface functionality. FIGURE 7-1 identifies the architectural components of the Netra CT software services.

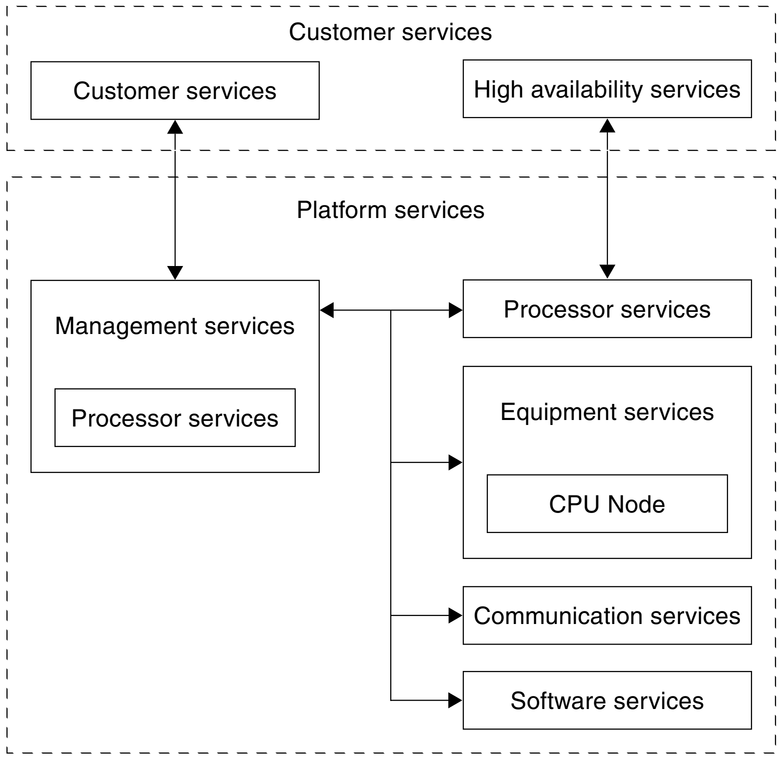


FIGURE 7-1 Netra CT Software Services

In a Netra CT cluster, the PMS software runs on both the alarm cards and the CPU boards. The PMS software running on alarm cards provides local and remote service connections for managing the CPU cards in its system. The PMS software running on CPU cards provides local and remote service connections for managing the resources running on the board, and the software provides remote access for managing resources running on other CPU cards in a PMS cluster.

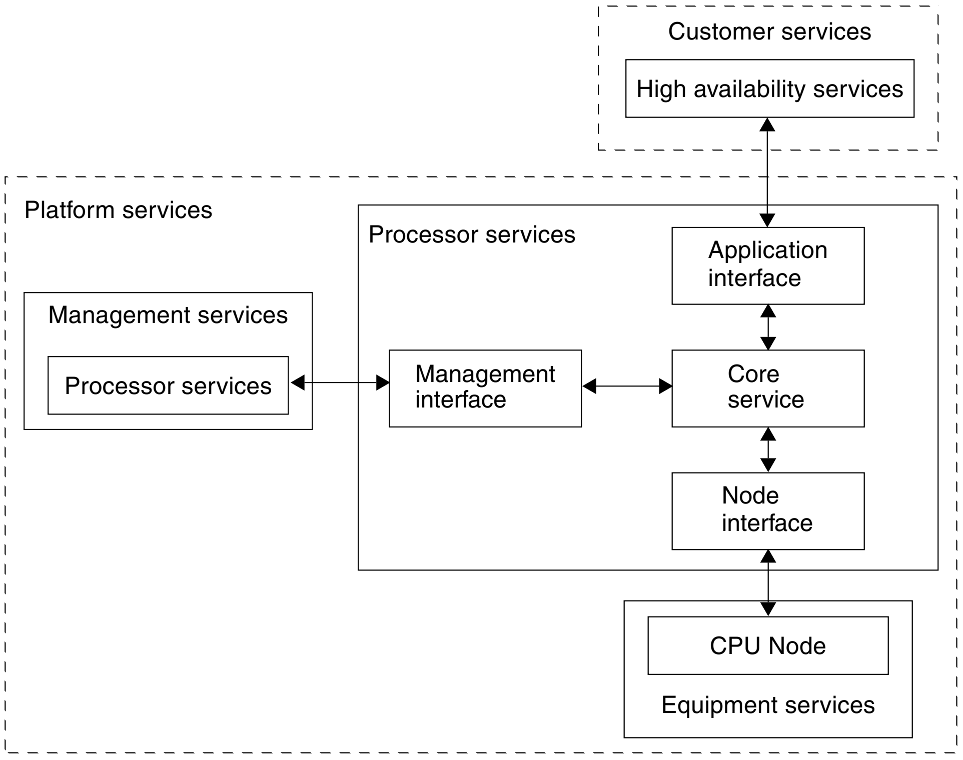


FIGURE 7-2 PMS Software Services and Interfaces

FIGURE 7-2 indicates the internal interfaces of the processor services.

The PMS software organizes the CPU resources it manages into the following three groups:

- Resource group 0 (RG0) – Specific application services
- Resource group 1 (RG1) – Operating system functionality
- Resource group 2 (RG2) – CPU hardware and the remaining processor board resources

The PMS software that runs on both alarm cards and the CPU cards divides its functionality along client-side and server-side (daemon-side) lines. The common client-side function provides a shared API for up to eight simultaneous application service processes. The core API functionality includes API control, PMS daemon control, application PMS connectivity, and application message send and receive with function execution. The API provides per-process serialization and separate threads for message reception and user-defined function execution, and messaging process timing.

In a typical example, a PMS client detects resource failures remotely and then remotely activates replacement resources such as those found in high-availability applications. The common daemon function provides server-side control and monitoring functionality for up to 16 remote CPUs. The daemon function also provides client-side functionality for controlling and monitoring up to 16 remote CPUs simultaneously with minimized latency by way of per-remote-CPU threading, as well as daemon control and performance monitoring and resource group monitoring and control.

From the client side, the alarm card function available by way of the send and receive messaging API is broken into *management* and *drawer* blocks. (The PMS software refers to Netra CT systems as drawers.) The CPU cards are divided into *management node* and *remote node drawer* (RND) views. The management view on both the alarm card and the CPU board provides administrative control and status over the PMS daemon as a whole. The management view also monitors the PMS software's performance.

The drawer (system) view by means of the alarm card provides the following administrative controls and monitors of the RG2 (hardware) resources: Core power down, power up, and reset. For RG1 (operating system) resources, this view also provides the following administrative controls and monitors: core shutdown, boot, and reboot. For RG0 (application services), this view provides off-line and active administrative controls. Finally, for the combined resource groups, this view provides the following administrative controls and monitors: Core maintenance, and operational configuration, five recovery processes, and the graceful reboot of the group.

The node view, by way of the CPU card itself, provides a much reduced set of administrative controls and monitors relative to the drawer view of the hardware, operating system, and the same administrative controls and monitors of the application services. In RG2 only reset administrative controls exists, but no monitors. Likewise, in RG1 only reboot administrative controls exist, but no monitors. In this view, there is no administrative control over the combined resource groups.

The CPU card RND view provides remote system view administrative controls and monitors to all the resource groups, with the exception of an alarm card failure. In this failure case, a reduced remote node view is used.

The PMS software execution performance is targeted by scheduling optimizations as well as using lightweight, proprietary messaging protocols, intersystem data encoding, and packetization protocols. The PMS software scalability due to CPU card growth is addressed by a per-CPU multithreading of up to 16 remote CPU cards per CPU. Application client growth is addressed by way of per-process multi-threading with up to eight client processes per PMS daemon.

The PMS software performance and reliability in cluster communication is also addressed with a messaging infrastructure that supports unidirectional and bidirectional point-to-point and unidirectional point-multipoint channels. This infrastructure includes source time-stamping available to the client for latency detection, call and return time-out for failure detection, and interprocess and intersystem TCP/IP socket streams for connection control, reachability determination, and reliable transport.

PMS Man Pages

The PMS software application programming interface (API) has been documented completely in the UNIX man pages included with the Netra CT software. TABLE 7-1 lists the man pages included with the Netra CT PMS software:

TABLE 7-1 Processor Management Services Man Pages

Man page	Description
`pms(1M)`	Provides an overview of the PMS software.
`pmsd(1M)`	Describes how to start and stop the CPU board PMS daemon (`pmsd`) and lists the daemon’s command line options.
`pmsd_ac(1M)`	Describes how to start and stop the alarm card PMS daemon (`pmsd_ac`) from the command line interface, and lists all the daemon’s other command-line functions.
`pms_apistart(1M)`	Describes the PMS API functions used to initialize (`pms_apistart`) and to free up (`pms_apistop`) PMS API resources in a PMS process. The man page also documents the functions used to take PMS out of an inactive state (`pms_start`) and to return it to an inactive state (`pms_stop`).
`pms_connect(1M)`	Documents the PMS API functions used to create (`pms_connect`) and destroy (`pms_disconnect`) a PMS daemon interface session.

TABLE 7-1 Processor Management Services Man Pages *(Continued)*

Man page	Description
`pms_send(1M)`	Describes the PMS API functions that enable PMS clients to send (`pms_send`) and receive (`pms_receive`) messages with other PMS clients or clusters.
`pms_usermgmt_message_payloads(1M)`	Describes the payloads for the user and management PMS function groups.
`pms_node_message_payloads(1M)`	Defines the payloads for the node PMS function group.
`pms_rnd_message_payloads(1M)`	Describes the payloads for the remote node drawer (system) PMS function group.

If you cannot view these man pages, add the PMS man page directory location to your `$MANPATH` environment variable. By default, the PMS man pages are installed in the following directory: `/opt/SUNWnetract/mgmt2.0/man`. Depending on the UNIX shell you are using, this variable might be defined in a shell startup file. Refer to the Solaris documentation for instructions on adding the PMS man page directory to a UNIX shell startup file on your system.

PMS Examples

The following examples show how to initialize a PMS client, the structure of the main thread, asynchronous messaging, scheduling, and the PMS client's user and management, node, and RND interfaces.

- "PMS Client Initialization Example" on page 216
- "PMS Client Main Thread" on page 223
- "PMS Client Asynchronous Message Handling" on page 225
- "PMS Client Scheduling Example" on page 238
- "PMS Client User and Management Interface" on page 239
- "PMS Client Node Interface" on page 256
- "PMS Client RND Interface" on page 263

CODE EXAMPLE 7-1 begins by initializing the main thread for a PMS client.

CODE EXAMPLE 7-1 PMS Client Initialization Example

```
#include <sys/types.h>          /* socketpair() */
#include <sys/socket.h>         /* socketpair() */

#include <unistd.h>             /* write(), read() */
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
#include <signal.h>             /* sigemptyset(), sigaddset(), sigaction() */
#include <time.h>               /* timer_create(), timer_settime() */

#include <stdio.h>              /* printf(), scanf() */


#include "pms.h"


/*      Application State Machine Example Overview:
      1) PMS API initialization and usage.
      2) PMS Daemon connectivity and availability management.
      3) Named application synchronization and behavior.
      4) Remote Node Drawer address list synchronization and monitoring.
      5) Basic example data caching synchronization on the client side for PMS
         items a particular application's intent/design makes it interested in.
      6) Basic asynchronous message handling infrastructure for the application.
      7) Remote monitoring of remote node drawer's.
      8) Example Control of a pair of remote node drawer's(not implemented yet).
*/



void*   app_hasim_thread(void*);


/* Event message handlers.. */

/* This mechanism registers one receive handler with PMS for all messages, which
   simply posts the messages to the client thread's processing queue to have them
   handled synchronously.  Alternatively, handlers can be registered with PMS
   individually in which case they will execute asynchronous to the client thread
   in the context of the PMS API receive thread. */

void    app_hasim_receive_post(struct pms_receive *pr);
int     app_hasim_receive_dispatch(struct pms_receive* pr);

void    app_hasim_receive_user_status(struct pms_receive *pr);
void    app_hasim_receive_mgmt_status(struct pms_receive *pr);
void    app_hasim_receive_node_rg0_status(struct pms_receive *pr);
void    app_hasim_receive_node_rg0_app_state_set_execute\
           (struct pms_receive *pr);
void    app_hasim_receive_rnd_status(struct pms_receive *pr);
void    app_hasim_receive_rnd_md0_status(struct pms_receive *pr);

void    app_hasim_receive_time_status(void);
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
/* Convenient state machine process sub-groupings.. */

void    app_hasim_user_process(void);
void    app_hasim_mgmt_process(void);
void    app_hasim_node_process(void);
void    app_hasim_rnd_process(void);
void    app_hasim_process(void);

/* Timer signal handler.. */

void    app_hasim_sigusr1_signal_handler(int);


/* Interval's currently set for example convenience.. */

#define HASIM_CHECK_INTERVAL                                    2
#define HASIM_SYNCCHECK_INTERVAL                                600
#define HASIM_CHECK_VALID_INTERVAL                              1800
#define HASIM_CHECK_INVALID_INTERVAL                            3600

#define HASIM_RND_ADDRESS_AUDIT_ENTRYS                          2


struct hasim_info
        {
        int                                     sockfd[2];
        struct
                {
                char                            node_ip_address[20];
                char                            drawer_ip_address[20];
                int                             node_slot_number;
                } rnd_address[HASIM_RND_ADDRESS_AUDIT_ENTRYS];

        struct
                {
#define HASIM_USER_RECEIVE_UNREGISTERED                         0x00
#define HASIM_USER_RECEIVE_REGISTERED                           0x01
                int                             receive_state;
#define HASIM_USER_PMS_VIEW_REACHABLE                           0x00
#define HASIM_USER_PMS_VIEW_UNREACHABLE                         0x01
                int                             pms_view;

                int                             view_cache;
                } user_info;

        struct
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
                {
#define HASIM_MGMT_RECEIVE_UNREGISTERED                         0x00
#define HASIM_MGMT_RECEIVE_REGISTERED                           0x01
                int                             receive_state;
#define HASIM_MGMT_PMS_STATE_UNAVAILABLE                        0x00
#define HASIM_MGMT_PMS_STATE_AVAILABLE                          0x01
                int                             pms_state;
#define HASIM_MGMT_RND_ADDRESS_UNVERIFIED                       0x00
#define HASIM_MGMT_RND_ADDRESS_VERIFIED                         0x01
                int                             rnd_address_state;
                int                             rnd_address_identifier[16];

#define HASIM_MGMT_CACHE_INVALID                                0x00
#define HASIM_MGMT_CACHE_OLD                                    0x01
#define HASIM_MGMT_CACHE_VALID                                  0x02
                int                             cache_state;
                int                             last_update;
                int                             last_sync_check;

                int                             mgmt_state_cache;
                struct
                        {
                        int                     identifier;
                        char                    node_ip_address[20];
                        char                    drawer_ip_address[20];
                        int                     node_slot_number;
                        } rnd_address_cache[16];
                } mgmt_info;

        struct
                {
#define HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED                        0x02
#define HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED         0x04
#define HASIM_NODE_GROUP_RECEIVE_UNREGISTERED                           0x00
#define HASIM_NODE_GROUP_RECEIVE_REGISTERED                             0x06
                int                             receive_state;
#define HASIM_NODE_RG0_APP_NAME_UNREGISTERED                    0x00
#define HASIM_NODE_RG0_APP_NAME_REGISTERED                      0x01
                int                             rg0_app_name_state;
#define HASIM_NODE_SERVICE_STATE_OFFLINE                        0x00
#define HASIM_NODE_SERVICE_STATE_ACTIVE                         0x01
                int                             service_state;

#define HASIM_NODE_CACHE_INVALID                                0x00
#define HASIM_NODE_CACHE_OLD                                    0x01
#define HASIM_NODE_CACHE_VALID                                  0x02
                int                             cache_state;
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
                int                             last_update;
                int                             last_sync_check;

                int                             rg0_state_cache;
                } node_info;

        struct
                {
#define HASIM_RND_RECEIVE_REGISTERED                            0x01
#define HASIM_RND_MD0_RECEIVE_REGISTERED                        0x20
#define HASIM_RND_GROUP_RECEIVE_UNREGISTERED                    0x00
#define HASIM_RND_GROUP_RECEIVE_REGISTERED                      0x21
                int                             receive_state;

#define HASIM_RND_CACHE_INVALID                                 0x00
#define HASIM_RND_CACHE_OLD                                     0x01
#define HASIM_RND_CACHE_VALID                                   0x02
                int                             cache_state;
                int                             last_update;
                int                             last_sync_check;

                int                             view_cache;
                int                             md0_config_cache;
                } rnd_info[16];

        };

static struct hasim_info                        mdi;


int
main(int argc, char *argv[])
{

  struct pms_receive    pr;
  struct sigaction      sigusr1_signal_handler_info;
  struct sigevent       evp;
  timer_t               timerid;
  struct itimerspec     val;
  struct itimerspec     oval;

  int                   i;


  if (argc != 1)
    {
    printf("Invalid Arguments\n");
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
    exit(1);
    }


  /* Start/Initialize the PMS API before using any further calls.. */

  if (pms_apistart() == -1)
    exit(2);


  /* Create message queue.. */

  if (socketpair(AF_UNIX, SOCK_DGRAM, 0, mdi.sockfd) == -1)
    {
    exit(3);
    }







  /* Setup defaults.. */

  /* Audit DB hardcoding for this example.. */

  strcpy(&mdi.rnd_address[0].node_ip_address[0], "129.150.94.70");
  strcpy(&mdi.rnd_address[0].drawer_ip_address[0], "129.150.151.140");
  mdi.rnd_address[0].node_slot_number = 2;
  strcpy(&mdi.rnd_address[1].node_ip_address[0], "129.150.94.58");
  strcpy(&mdi.rnd_address[1].drawer_ip_address[0], "129.150.151.143");
  mdi.rnd_address[1].node_slot_number = 3;


  mdi.user_info.receive_state = HASIM_USER_RECEIVE_UNREGISTERED;
  mdi.user_info.pms_view = HASIM_USER_PMS_VIEW_UNREACHABLE;

  mdi.mgmt_info.receive_state = HASIM_MGMT_RECEIVE_UNREGISTERED;
  mdi.mgmt_info.pms_state = HASIM_MGMT_PMS_STATE_UNAVAILABLE;
  mdi.mgmt_info.rnd_address_state = HASIM_MGMT_RND_ADDRESS_UNVERIFIED;
  for(i=0;i<16;i++)
    mdi.mgmt_info.rnd_address_identifier[i] = -1;
  mdi.mgmt_info.cache_state = HASIM_MGMT_CACHE_INVALID;
  mdi.mgmt_info.last_update = HASIM_CHECK_INVALID_INTERVAL;
  mdi.mgmt_info.last_sync_check = HASIM_SYNCCHECK_INTERVAL;
```

CODE EXAMPLE 7-1 PMS Client Initialization Example *(Continued)*

```
  mdi.node_info.receive_state = HASIM_NODE_GROUP_RECEIVE_UNREGISTERED;
  mdi.node_info.rg0_app_name_state = HASIM_NODE_RG0_APP_NAME_UNREGISTERED;
  mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
  mdi.node_info.cache_state = HASIM_NODE_CACHE_INVALID;
  mdi.node_info.last_update = HASIM_CHECK_INVALID_INTERVAL;
  mdi.node_info.last_sync_check = HASIM_SYNCCHECK_INTERVAL;

  for(i=0;i<16;i++)
    {
    mdi.rnd_info[i].receive_state = HASIM_RND_GROUP_RECEIVE_UNREGISTERED;
    mdi.rnd_info[i].cache_state = HASIM_RND_CACHE_INVALID;
    mdi.rnd_info[i].last_update = HASIM_CHECK_INVALID_INTERVAL;
    mdi.rnd_info[i].last_sync_check = HASIM_SYNCCHECK_INTERVAL;
    }


  /* Setup timer.. */

  sigemptyset(&sigusr1_signal_handler_info.sa_mask);
  sigaddset(&sigusr1_signal_handler_info.sa_mask, SIGUSR1);
  sigusr1_signal_handler_info.sa_flags = 0;
  sigusr1_signal_handler_info.sa_handler = app_hasim_sigusr1_signal_handler;
  sigaction(SIGUSR1, &sigusr1_signal_handler_info, NULL);

  evp.sigev_notify = SIGEV_SIGNAL;
  evp.sigev_signo = SIGUSR1;

  if (timer_create(CLOCK_REALTIME, &evp, &timerid) == -1)
    exit(4);

  val.it_value.tv_sec = HASIM_CHECK_INTERVAL;
  val.it_value.tv_nsec = 0;
  val.it_interval.tv_sec = HASIM_CHECK_INTERVAL;
  val.it_interval.tv_nsec = 0;

  if (timer_settime(timerid, TIMER_RELTIME,  &val, NULL) == -1)
    exit(4);


  /* Don't bother creating another thread, run in context of main default.. */
  app_hasim_thread(0);


}
```

CODE EXAMPLE 7-2 PMS Client Main Thread

```
void*
app_hasim_thread(void* arg)
{

  char                  receivebuffer[256];
  int                   receivestatus;

  fd_set                readfds;
  int                   select_return;
  struct timeval        timeout;

  struct pms_send       ps;
  struct pms_receive    pr;

  int                   i;


  printf("*** HA Client Application Simulation ***\n");


  /* Presuming PMS will have been started at boot or by another app.. */


  timeout.tv_sec = HASIM_CHECK_INTERVAL;
  timeout.tv_usec = 0;


  while(1)
    {

    FD_ZERO(&readfds);

    FD_SET(mdi.sockfd[1], &readfds);

    /* Wait for event messages.. */

    select_return = select(64, &readfds, NULL, NULL, &timeout);

    if (select_return > 0)
      {
      if (FD_ISSET(mdi.sockfd[1], &readfds) != 0)
        {

        receivestatus = read(mdi.sockfd[1], &receivebuffer[0], 256);
```

CODE EXAMPLE 7-2 PMS Client Main Thread *(Continued)*

```
        if (receivestatus <= 0)
          {

          /* Handle Error.. */

          }
        else
          {

          /* Handle Message.. */

          app_hasim_receive_dispatch((struct pms_receive*)&receivebuffer[0]);

          }
        }
      }
    else if (select_return == 0)
      {

      /* Handle Timeout.. */

      }
    else
      {

      /* Handle Error.. */

      }

    }


}


void
app_hasim_sigusr1_signal_handler(int signal)
{

  struct pms_receive    pr;


  pr.session.type = PMS_SR_CALL_NO_RETURN;
  pr.payload.type = PMS_PD_PAYLOAD_TYPE_MAX+1;
```

CODE EXAMPLE 7-2 PMS Client Main Thread *(Continued)*

```
   app_hasim_receive_post(&pr);

 }
```

The following example sets up a PMS client to handle asynchronous messages.

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling

```
void
app_hasim_receive_post(struct pms_receive* pr)
{


  int                   status;


  /* Write for reading in context of main thread.. */

  status = write(mdi.sockfd[0], pr, sizeof(struct pms_receive));

  if (status < 0)
    {
    }


}



int
app_hasim_receive_dispatch(struct pms_receive* pr)
{


  switch(pr->payload.type)
    {
    case PMS_PD_USER_STATUS:

      app_hasim_receive_user_status(pr);

    break;

    case PMS_PD_MGMT_STATUS:

      app_hasim_receive_mgmt_status(pr);
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
    break;

    case PMS_PD_NODE_RG0_STATUS:

      app_hasim_receive_node_rg0_status(pr);

    break;

    case PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE:

      app_hasim_receive_node_rg0_app_state_set_execute(pr);

    break;
    case PMS_PD_RND_STATUS:

      app_hasim_receive_rnd_status(pr);

    break;
    case PMS_PD_RND_MD0_STATUS:

      app_hasim_receive_rnd_md0_status(pr);

    break;
    case PMS_PD_PAYLOAD_TYPE_MAX+1:

      app_hasim_receive_time_status();

    break;
    }


  return(0);

}


void
app_hasim_receive_user_status(struct pms_receive* pr)
{


  switch(pr->payload.data.user_status.code)
    {
    case PMS_PD_USER_STATUS_PMS_REACHABLE:

      printf("hasim :        received USER_STATUS PMS_REACHABLE..\n");
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      mdi.user_info.view_cache = PMS_PD_USER_STATUS_PMS_REACHABLE;

      /* Run state machine.. */

      app_hasim_process();

    break;
    case PMS_PD_USER_STATUS_PMS_UNREACHABLE:

      printf("hasim :        received USER_STATUS PMS_UNREACHABLE..\n");


      mdi.user_info.view_cache = PMS_PD_USER_STATUS_PMS_UNREACHABLE;


      app_hasim_process();

    break;
    }


}



void
app_hasim_receive_mgmt_status(struct pms_receive* pr)
{

  struct pms_send       ps;
  struct pms_receive    prs;

  int                   info_get_fail;

  int                   rnd_address_identifier[16];
  char                  rnd_address_node_ip_address[16][20];
  char                  rnd_address_drawer_ip_address[16][20];
  int                   rnd_address_node_slot_number[16];

  int                   i, j;


  switch(pr->payload.data.mgmt_status.code)
    {
    case PMS_PD_MGMT_STATUS_PMS_STATE_AVAILABLE:
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      printf("hasim :        received MGMT_STATUS PMS STATE AVAILABLE..\n");

      /* Update cached data and set update time.. */

      mdi.mgmt_info.mgmt_state_cache = PMS_PD_MGMT_INFO_GET_STATUS_AVAILABLE;
      mdi.mgmt_info.last_update = 0;

      app_hasim_process();

    break;
    case PMS_PD_MGMT_STATUS_PMS_STATE_UNAVAILABLE:

      printf("hasim :        received MGMT_STATUS PMS STATE UNAVAILABLE..\n");

     mdi.mgmt_info.mgmt_state_cache = PMS_PD_MGMT_INFO_GET_STATUS_UNAVAILABLE;
      mdi.mgmt_info.last_update = 0;

      app_hasim_process();

    break;
    case PMS_PD_MGMT_STATUS_PMS_ADMIN_STATE_FORCE_UNAVAILABLE:

      printf("hasim :        received MGMT_STATUS PMS ADMIN STATE FORCE\
         UNAVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_MGMT_STATUS_PMS_ADMIN_STATE_VOTE_AVAILABLE:

     printf("hasim :       received MGMT_STATUS PMS ADMIN STATE VOTE AVAILABLE\
        ..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_MGMT_STATUS_PMS_ADMIN_STATE_FORCE_AVAILABLE:

      printf("hasim :        received MGMT_STATUS PMS ADMIN STATE FORCE \
        AVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_MGMT_STATUS_PMS_PERFORMANCE_DEGRADED:
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      printf("hasim :       received MGMT_STATUS PMS PERFORMANCE DEGRADED..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_MGMT_STATUS_RND_ADDRESS_ADD:
     case PMS_PD_MGMT_STATUS_RND_ADDRESS_DELETE:

       if (pr->payload.data.mgmt_status.code == \
         PMS_PD_MGMT_STATUS_RND_ADDRESS_ADD)
         printf("hasim :        received MGMT_STATUS RND ADDRESS ADD..\n");
       else
         printf("hasim :        received MGMT_STATUS RND ADDRESS DELETE..\n");


       info_get_fail = 0;


       /* Get MGMT rnd address information.. */

       ps.session.type = PMS_SR_CALL_RETURN_TIMED;
       ps.session.info.crt.time = 0;
       ps.payload.type = PMS_PD_MGMT_RND_ADDRESS_INFO_GET_EXECUTE;

       for(i=0;i<16;i++)
         {
         ps.payload.data.mgmt_rnd_address_info_get_execute.index = i;

         if (pms_send(&ps, &prs) == 0)
           {
           if (prs.payload.data.mgmt_rnd_address_info_get_status.err == \
             PMS_PD_MGMT_RND_ADDRESS_INFO_GET_STATUS_ERR_NONE)
             {
             rnd_address_identifier[i] = \
               prs.payload.data.mgmt_rnd_address_info_get_status.identifier;
             strncpy(&rnd_address_node_ip_address[i][0], \

&prs.payload.data.mgmt_rnd_address_info_get_status.node_ip_address[0], 20);
             strncpy(&rnd_address_drawer_ip_address[i][0], \

&prs.payload.data.mgmt_rnd_address_info_get_status.drawer_ip_address[0], 20);
             rnd_address_node_slot_number[i] = \
            prs.payload.data.mgmt_rnd_address_info_get_status.node_slot_number;
             }
           else
             {
             info_get_fail = 1;
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
            }
          }
        else
          {
          info_get_fail = 1;
          }
        }


      if (info_get_fail == 0)
        {







        for(i=0;i<16;i++)
          {
          mdi.mgmt_info.rnd_address_cache[i].identifier = \
            rnd_address_identifier[i];
          strncpy(&mdi.mgmt_info.rnd_address_cache[i].node_ip_address[0], \
            &rnd_address_node_ip_address[i][0], 20);
          strncpy(&mdi.mgmt_info.rnd_address_cache[i].drawer_ip_address[0], \
            &rnd_address_drawer_ip_address[i][0], 20);
          mdi.mgmt_info.rnd_address_cache[i].node_slot_number = \
            rnd_address_node_slot_number[i];
          }

        mdi.mgmt_info.last_update = 0;
        }


      app_hasim_process();

    break;
    case PMS_PD_MGMT_STATUS_PMS_ADMIN_STATE_AV_RG0VA_DELAY:

      printf("hasim :        received MGMT_STATUS PMS ADMIN STATE AV RG0VA \
         DELAY..\n");

      /* Doing nothing at the moment.. */

    break;
    }

}
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
void
app_hasim_receive_node_rg0_status(struct pms_receive* pr)
{


  switch(pr->payload.data.node_rg0_status.code)
    {
    case PMS_PD_NODE_RG0_STATUS_STATE_ACTIVE:

      printf("hasim :        received NODE_RG0_STATUS STATE ACTIVE..\n");

      mdi.node_info.rg0_state_cache = PMS_PD_NODE_RG0_INFO_GET_STATUS_ACTIVE;
      mdi.node_info.last_update = 0;

      app_hasim_process();

    break;
    case PMS_PD_NODE_RG0_STATUS_STATE_OFFLINE:

      printf("hasim :        received NODE_RG0_STATUS STATE OFFLINE..\n");

      mdi.node_info.rg0_state_cache = PMS_PD_NODE_RG0_INFO_GET_STATUS_OFFLINE;
      mdi.node_info.last_update = 0;

      app_hasim_process();

    break;
    case PMS_PD_NODE_RG0_STATUS_ADMIN_STATE_FORCE_OFFLINE:

      printf("hasim :        received NODE_RG0_STATUS ADMIN STATE FORCE \
         OFFLINE ..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_NODE_RG0_STATUS_ADMIN_STATE_VOTE_ACTIVE:

      printf("hasim :        received NODE_RG0_STATUS ADMIN STATE VOTE \
         ACTIVE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_NODE_RG0_STATUS_ADMIN_STATE_FORCE_ACTIVE:
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      printf("hasim :        received NODE_RG0_STATUS ADMIN STATE FORCE \
         ACTIVE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_NODE_RG0_STATUS_APP_STATE_SET_FAULT:

     printf("hasim :       received NODE_RG0_STATUS APP STATE SET FAULT..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_NODE_RG0_STATUS_ADOPER_STATUSMASK_SET:

     printf("hasim :       received NODE_RG0_STATUS ADOPER STATUSMASK SET..\n");

      /* Doing nothing at the moment.. */

    break;
    }

}



void
app_hasim_receive_node_rg0_app_state_set_execute(struct pms_receive* pr)
{

  struct pms_send               ps;


  switch(pr->payload.data.node_rg0_app_state_set_execute.state)
    {
    case PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE_ACTIVE:

     printf("hasim :       received NODE_RG0_APP_STATE_SET_EXECUTE ACTIVE..\n");


      /* Do whatever, within pr->session.info.crt.time if possible.. */


      /* Send return message indicating successful reception.. */

      ps.session.type = PMS_SR_RETURN;
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      ps.session.info.r.return_identifier = \
        pr->session.info.crt.call_identifier;
     ps.session.info.r.return_priority = pr->session.info.crt.return_priority;

      ps.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_STATUS;
      ps.payload.data.node_rg0_app_state_set_status.err = \
        PMS_PD_NODE_RG0_APP_STATE_SET_STATUS_SUCCESS;


      if (pms_send(&ps, 0) != 0)
        {
        }

    break;
    case PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE_OFFLINE:

      printf("hasim :        received NODE_RG0_APP_STATE_SET_EXECUTE OFFLINE\
         ..\n");


      /* Do whatever, within pr->session.info.crt.time if possible.. */


      ps.session.type = PMS_SR_RETURN;
      ps.session.info.r.return_identifier = \
          pr->session.info.crt.call_identifier;
     ps.session.info.r.return_priority = pr->session.info.crt.return_priority;

      ps.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_STATUS;
      ps.payload.data.node_rg0_app_state_set_status.err = \
        PMS_PD_NODE_RG0_APP_STATE_SET_STATUS_SUCCESS;


      if (pms_send(&ps, 0) != 0)
        {
        }

    break;
    };

}



void
app_hasim_receive_rnd_status(struct pms_receive* pr)
{
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
  printf("hasim :        rs.identifier=%.8X\n", \
     pr->payload.data.rnd_status.identifier);


  switch(pr->payload.data.rnd_status.code)
    {
    case PMS_PD_RND_STATUS_VIEW_NODE_REACHABLE_DRAWER_REACHABLE:

      printf("hasim :        received RND_STATUS NODE REACHABLE DRAWER \
         REACHABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_VIEW_NODE_REACHABLE_DRAWER_UNREACHABLE:

      printf("hasim :        received RND_STATUS NODE REACHABLE DRAWER\
         UNREACHABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_VIEW_NODE_UNREACHABLE_DRAWER_REACHABLE:

      printf("hasim :        received RND_STATUS NODE UNREACHABLE DRAWER\
         REACHABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_VIEW_NODE_UNREACHABLE_DRAWER_UNREACHABLE:

      printf("hasim :        received RND_STATUS NODE UNREACHABLE DRAWER\
         UNREACHABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_ADOPER_FORCE_UNAVAILABLE:

     printf("hasim :        received RND_STATUS ADOPER FORCE UNAVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
    case PMS_PD_RND_STATUS_ADOPER_VOTE_AVAILABLE:

      printf("hasim :        received RND_STATUS ADOPER VOTE AVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_ADOPER_FORCE_AVAILABLE:

      printf("hasim :        received RND_STATUS ADOPER FORCE AVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_ADOPER_STATUSMASK_SET:

      printf("hasim :        received RND_STATUS ADOPER STATUSMASK SET..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_STATE_UNAVAILABLE:

      printf("hasim :        received RND_STATUS STATE UNAVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_STATUS_STATE_AVAILABLE:

      printf("hasim :        received RND_STATUS STATE AVAILABLE..\n");

      /* Doing nothing at the moment.. */

    break;
    }

}



void
app_hasim_receive_rnd_md0_status(struct pms_receive* pr)
{


  printf("hasim :        rms.identifier=%.8X\n", \
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      pr->payload.data.rnd_md0_status.identifier);


   switch(pr->payload.data.rnd_md0_status.code)
     {
     case PMS_PD_RND_MD0_STATUS_ADOPER_CONFIG_MAINTENANCE:

       printf("hasim :        received RND_MD0_STATUS ADOPER CONFIG \
          MAINTENANCE..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_RND_MD0_STATUS_ADOPER_CONFIG_OPERATIONAL:

       printf("hasim :        received RND_MD0_STATUS ADOPER CONFIG \
          OPERATIONAL..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_RND_MD0_STATUS_ADOPER_GRACEFUL_REBOOT:

      printf("hasim :       received RND_MD0_STATUS ADOPER GRACEFUL REBOOT..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_RND_MD0_STATUS_ADOPER_STATUSMASK_SET:

      printf("hasim :       received RND_MD0_STATUS ADOPER STATUSMASK SET..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERY_PC:

       printf("hasim :        received RND_MD0_STATUS ADOPER RECOVERY PC..\n");

       /* Doing nothing at the moment.. */

     break;
     case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERY_RST:

       printf("hasim :        received RND_MD0_STATUS ADOPER RECOVERY RST..\n");

       /* Doing nothing at the moment.. */
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
    break;
    case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERY_RSTPC:

     printf("hasim :       received RND_MD0_STATUS ADOPER RECOVERY RSTPC..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERY_PD:
      printf("hasim :        received RND_MD0_STATUS ADOPER RECOVERY PD..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERY_RB:

      printf("hasim :        received RND_MD0_STATUS ADOPER RECOVERY RB..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_ADOPER_RECOVERYAUTOMODE_SET:

      printf("hasim :        received RND_MD0_STATUS ADOPER RECOVERYAUTOMODE\
         SET..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_ADOPER_SCDM_TIMEOUT:

      printf("hasim :       received RND_MD0_STATUS ADOPER SCDM TIMEOUT..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_CONFIG_MAINTENANCE:

      printf("hasim :        received RND_MD0_STATUS CONFIG MAINTENANCE..\n");

      /* Doing nothing at the moment.. */

    break;
    case PMS_PD_RND_MD0_STATUS_CONFIG_OPERATIONAL:

      printf("hasim :        received RND_MD0_STATUS CONFIG OPERATIONAL..\n");
```

CODE EXAMPLE 7-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      /* Doing nothing at the moment.. */

    break;
    }

}
```

The following example shows a PMS client's scheduling.

CODE EXAMPLE 7-4 PMS Client Scheduling Example

```
void
app_hasim_receive_time_status(void)
{

  int                  i;
  mdi.mgmt_info.last_update += HASIM_CHECK_INTERVAL;
  mdi.mgmt_info.last_sync_check += HASIM_CHECK_INTERVAL;

  mdi.node_info.last_update += HASIM_CHECK_INTERVAL;
  mdi.node_info.last_sync_check += HASIM_CHECK_INTERVAL;

  for(i=0;i<16;i++)
    {
    mdi.rnd_info[i].last_update += HASIM_CHECK_INTERVAL;
    mdi.rnd_info[i].last_sync_check += HASIM_CHECK_INTERVAL;
    }


  app_hasim_process();


}



void
app_hasim_process(void)
{


  /* Run state machine sub-groupings.. */
```

CODE EXAMPLE 7-4 PMS Client Scheduling Example *(Continued)*

```
  app_hasim_user_process();

  app_hasim_mgmt_process();

  app_hasim_node_process();

  app_hasim_rnd_process();


}
```

The following example shows the PMS client's user management interface.

CODE EXAMPLE 7-5 PMS Client User and Management Interface

```
void
app_hasim_user_process(void)
{

  struct pms_receive    pr;

  int                   i;





  /* PMS View check */

  /* Periodically attempt to connect if unreachable. Return to initial
     state variable settings on reachable to unreachable transition.. */

  if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_UNREACHABLE)
    {
    if (pms_connect(PMS_SERVER_PORT_NUMBER_DEFAULT) != 0)
      {
      }
    else
      {
      mdi.user_info.pms_view = HASIM_USER_PMS_VIEW_REACHABLE;
      mdi.user_info.view_cache = PMS_PD_USER_STATUS_PMS_REACHABLE;
      }
    }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
   else /* mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE */
     {
     if (mdi.user_info.view_cache == PMS_PD_USER_STATUS_PMS_UNREACHABLE)
       {

       /* RND */

       for(i=0;i<16;i++)
         {

         mdi.rnd_info[i].cache_state = HASIM_RND_CACHE_INVALID;
         mdi.rnd_info[i].last_update = HASIM_CHECK_INVALID_INTERVAL;

        if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED) != 0)
           {
           pr.session.type = PMS_SR_CALL_NO_RETURN;
           pr.payload.type = PMS_PD_RND_STATUS;
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
           }

       if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
               != 0)
           {
           pr.session.type = PMS_SR_CALL_NO_RETURN;
           pr.payload.type = PMS_PD_RND_MD0_STATUS;
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
  }

         }


       /* NODE */

       mdi.node_info.cache_state = HASIM_NODE_CACHE_INVALID;
       mdi.node_info.last_update = HASIM_CHECK_INVALID_INTERVAL;

       if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
         {
         mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
         }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
       if (mdi.node_info.rg0_app_name_state ==\
             HASIM_NODE_RG0_APP_NAME_REGISTERED)
         {
        mdi.node_info.rg0_app_name_state = HASIM_NODE_RG0_APP_NAME_UNREGISTERED;
         }

       if ((mdi.node_info.receive_state &\
              HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED) != 0)
         {
         pr.session.type = PMS_SR_CALL_NO_RETURN;
         pr.payload.type = PMS_PD_NODE_RG0_STATUS;
         pms_receive(&pr, 0, 0);
         mdi.node_info.receive_state &=\
           !HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED;
         }

       if ((mdi.node_info.receive_state & \
         HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED) != 0)
         {
         pr.session.type = PMS_SR_CALL_RETURN_TIMED;
         pr.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE;
         pms_receive(&pr, 0, 0);
         mdi.node_info.receive_state &= \
           !HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED;
         }


       /* MGMT */

       mdi.mgmt_info.cache_state = HASIM_MGMT_CACHE_INVALID;
       mdi.mgmt_info.last_update = HASIM_CHECK_INVALID_INTERVAL;

       for(i=0;i<16;i++)
         mdi.mgmt_info.rnd_address_identifier[i] = -1;

       if (mdi.mgmt_info.rnd_address_state == HASIM_MGMT_RND_ADDRESS_VERIFIED)
         {
         mdi.mgmt_info.rnd_address_state = HASIM_MGMT_RND_ADDRESS_UNVERIFIED;
         }

       if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE)
         {
         mdi.mgmt_info.pms_state = HASIM_MGMT_PMS_STATE_UNAVAILABLE;
         }

       if (mdi.mgmt_info.receive_state == HASIM_MGMT_RECEIVE_REGISTERED)
         {
```

```
        pr.session.type = PMS_SR_CALL_NO_RETURN;
        pr.payload.type = PMS_PD_MGMT_STATUS;
        pms_receive(&pr, 0, 1);

        mdi.mgmt_info.receive_state = HASIM_MGMT_RECEIVE_UNREGISTERED;
        }


      /* USER */

      if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
        {
        pr.session.type = PMS_SR_CALL_NO_RETURN;
        pr.payload.type = PMS_PD_USER_STATUS;
        pms_receive(&pr, 0, 0);

        mdi.user_info.receive_state = HASIM_USER_RECEIVE_UNREGISTERED;
        }

      if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
        {
        pms_disconnect();

        mdi.user_info.pms_view = HASIM_USER_PMS_VIEW_UNREACHABLE;
        }

      }
    }


  /* Receive Check */

  /* If USER messages are not receive registered, attempt to register if PMS
     is reachable.. */

  if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
    {
    if (mdi.user_info.receive_state != HASIM_USER_RECEIVE_REGISTERED)
      {
      pr.session.type = PMS_SR_CALL_NO_RETURN;
      pr.payload.type = PMS_PD_USER_STATUS;
      if (pms_receive(&pr, app_hasim_receive_post, 0) != -1)
        mdi.user_info.receive_state = HASIM_USER_RECEIVE_REGISTERED;
      }
    }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
}



void
app_hasim_mgmt_process(void)
{

  struct pms_send       ps;
  struct pms_receive    pr;

  int                   info_get_fail;

  int                   match[HASIM_RND_ADDRESS_AUDIT_ENTRYS];

  int                   mgmt_state;
  int                   rnd_address_identifier[16];
  char                  rnd_address_node_ip_address[16][20];
  char                  rnd_address_drawer_ip_address[16][20];
  int                   rnd_address_node_slot_number[16];

  int                   i, j;


  /* Receive Check */

  /* If MGMT messages are not receive registered, attempt to register if PMS
     is reachable and USER receive messages are registered.  If registration
     is successful, force an initial cache update.. */

  if (mdi.mgmt_info.receive_state != HASIM_MGMT_RECEIVE_REGISTERED)
    {
    if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
      {
      if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
        {
        pr.session.type = PMS_SR_CALL_NO_RETURN;
        pr.payload.type = PMS_PD_MGMT_STATUS;
        if (pms_receive(&pr, app_hasim_receive_post, 1) != -1)
          mdi.mgmt_info.receive_state = HASIM_MGMT_RECEIVE_REGISTERED;

        /* Force an info_get immediately after registering.. */
        mdi.mgmt_info.last_sync_check = HASIM_SYNCCHECK_INTERVAL;
        }
      }
    }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
/* PMS State check */

/* Process PMS state transitions.  On an available to unavailable transition
   return to pre-NODE and RND operational state variable settings.. */

if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_UNAVAILABLE)
  {
  if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_INVALID)
    {
    if (mdi.mgmt_info.mgmt_state_cache != \
          PMS_PD_MGMT_INFO_GET_STATUS_UNAVAILABLE)
      {
      mdi.mgmt_info.pms_state = HASIM_MGMT_PMS_STATE_AVAILABLE;
      }
    }
  }
else /* mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE */
  {
  if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_INVALID)
    {
    if (mdi.mgmt_info.mgmt_state_cache == \
          PMS_PD_MGMT_INFO_GET_STATUS_UNAVAILABLE)
      {

      /* RND */

      for(i=0;i<16;i++)
        {

        if ((mdi.rnd_info[i].receive_state & \
              HASIM_RND_RECEIVE_REGISTERED) != 0)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_RND_STATUS;
          pr.payload.data.rnd_status.identifier = \
            mdi.mgmt_info.rnd_address_identifier[i];
          pms_receive(&pr, 0, 0);
          mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
          }





     if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
              != 0)
           {
           pr.session.type = PMS_SR_CALL_NO_RETURN;
           pr.payload.type = PMS_PD_RND_MD0_STATUS;
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
          mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
           }

         }


       /* NODE */

     if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
       {
       mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
       }

     if (mdi.node_info.rg0_app_name_state == \
           HASIM_NODE_RG0_APP_NAME_REGISTERED)
       {
      mdi.node_info.rg0_app_name_state = HASIM_NODE_RG0_APP_NAME_UNREGISTERED;
       }


     if ((mdi.node_info.receive_state &\
            HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED) != 0)
       {
       pr.session.type = PMS_SR_CALL_NO_RETURN;
       pr.payload.type = PMS_PD_NODE_RG0_STATUS;
       pms_receive(&pr, 0, 0);
       mdi.node_info.receive_state &=\
         !HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED;
       }

     if ((mdi.node_info.receive_state & \
       HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED) != 0)
       {
       pr.session.type = PMS_SR_CALL_RETURN_TIMED;
       pr.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE;
       pms_receive(&pr, 0, 0);
       mdi.node_info.receive_state &= \
         !HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED;
       }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
         mdi.mgmt_info.pms_state = HASIM_MGMT_PMS_STATE_UNAVAILABLE;
         }
       }
     }



   /* RND Address Check */

   /* Check once at startup if the RND address pairs currently in the list
      are the same as this control application's defaults.  If not, remove
      any that differ and add any that are missing.  This is a bit contrived
      to demonstrate interaction via the address list messages.  No point
      in starting processing if cache is invalid and PMS is not reachable
      and USER registration is not completed.. */

   if (mdi.mgmt_info.rnd_address_state != HASIM_MGMT_RND_ADDRESS_VERIFIED)
     {
     if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_INVALID)
       {
       if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
         {
         if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
           {

           match[0] = 0;
           match[1] = 0;

          /* Search RND address list for entries not in the app's verify list.. */

           for(i=0;i<16;i++)
             {
             if (mdi.mgmt_info.rnd_address_cache[i].identifier != -1)
               {
               for(j=0;j<HASIM_RND_ADDRESS_AUDIT_ENTRYS;j++)
                 {
                 if (match[j] == 0)
                   {
                   /* Use strcmp() for the moment. Use sockaddr_in when \
                      I get around to it.. */
                   if \
            (strcmp(&mdi.mgmt_info.rnd_address_cache[i].node_ip_address[0], \
                     &mdi.rnd_address[j].node_ip_address[0]) == 0)
                     {
                     if\
            (strcmp(&mdi.mgmt_info.rnd_address_cache[i].drawer_ip_address[0], \
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
                    &mdi.rnd_address[j].drawer_ip_address[0]) == 0)
                    {
                 if (mdi.mgmt_info.rnd_address_cache[i].node_slot_number == \
                      mdi.rnd_address[j].node_slot_number)
                      {
                      match[j] = 1;

                      break;
                      }
                    }
                  }
                }
              }

            /* Delete entries not in the app's verify list.. */

            if (j == HASIM_RND_ADDRESS_AUDIT_ENTRYS)
              {
              ps.session.type = PMS_SR_CALL_RETURN_TIMED;
              ps.session.info.crt.time = 0;
              ps.payload.type = PMS_PD_MGMT_RND_ADDRESS_DELETE_EXECUTE;

              ps.payload.data.mgmt_rnd_address_delete_execute.identifier = \
                mdi.mgmt_info.rnd_address_cache[i].identifier;

              if (pms_send(&ps, &pr) == 0)
                {
                if (pr.payload.data.mgmt_rnd_address_delete_status.err == \
                  PMS_PD_MGMT_RND_ADDRESS_DELETE_STATUS_ERR_NONE)
                  {
                  }
                }
              }

            }
          }

        /* Add any missing entries.. */

        for(i=0;i<HASIM_RND_ADDRESS_AUDIT_ENTRYS;i++)
          {
          if (match[i] == 0)
            {
            ps.session.type = PMS_SR_CALL_RETURN_TIMED;
            ps.session.info.crt.time = 0;
            ps.payload.type = PMS_PD_MGMT_RND_ADDRESS_ADD_EXECUTE;
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
 strncpy(&ps.payload.data.mgmt_rnd_address_add_execute.node_ip_address[0], \
               &mdi.rnd_address[i].node_ip_address[0], 20);
 strncpy(&ps.payload.data.mgmt_rnd_address_add_execute.drawer_ip_address[0], \
               &mdi.rnd_address[i].drawer_ip_address[0], 20);
            ps.payload.data.mgmt_rnd_address_add_execute.node_slot_number = \
               mdi.rnd_address[i].node_slot_number;

             if (pms_send(&ps, &pr) == 0)
               {
               if (pr.payload.data.mgmt_rnd_address_add_status.err == \
                 PMS_PD_MGMT_RND_ADDRESS_ADD_STATUS_ERR_NONE)
                 {
                 }
               }
             }
           }


         mdi.mgmt_info.rnd_address_state = HASIM_MGMT_RND_ADDRESS_VERIFIED;
         }
       }
     }
   }


  /* RND Address Identifier check */

  /* Process RND address identifier transitions.  On in-use to not-in-use
     transitions, return state variables to pre-RND initialized state for that
     identifier.  Check whether any list entries have been deleted and re-added
     since last processing and do an available->unavailable->available
     transition.. */

  for(i=0;i<16;i++)
    {

    if (mdi.mgmt_info.rnd_address_identifier[i] == -1)
      {

      if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_INVALID)
        {
        if (mdi.mgmt_info.rnd_address_cache[i].identifier != -1)
          {
          mdi.mgmt_info.rnd_address_identifier[i] =\
             mdi.mgmt_info.rnd_address_cache[i].identifier;
          }
        }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
      }
    else /* mdi.mgmt_info.rnd_address_identifier[i] != -1 */
      {

      if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_INVALID)
        {

        if (mdi.mgmt_info.rnd_address_cache[i].identifier == -1)
          {

          /* RND */

          if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED)\
               != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
            }

       if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
               != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_MD0_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
            }


          mdi.mgmt_info.rnd_address_identifier[i] = -1;
          }
        else
          {
          if (mdi.mgmt_info.rnd_address_identifier[i] != \
            mdi.mgmt_info.rnd_address_cache[i].identifier)
            {
            /* RND */

           if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED)\
                  != 0)
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
            }

          if ((mdi.rnd_info[i].receive_state & \
                 HASIM_RND_MD0_RECEIVE_REGISTERED) != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_MD0_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
          mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
            }

          mdi.mgmt_info.rnd_address_identifier[i] =\
             mdi.mgmt_info.rnd_address_cache[i].identifier;
          }
        }

      }

    }

  }


/* Sync Check */

/* Policy: Sync update checked every SYNCCHECK_INTERVAL seconds.. */
if (mdi.mgmt_info.last_sync_check > HASIM_SYNCCHECK_INTERVAL)
  {
  /* Policy:  Don't attempt a sync update if any async partial updates have
     been received within SYNCCHECK_INTERVAL.. */
  if (mdi.mgmt_info.last_update > HASIM_SYNCCHECK_INTERVAL)
    {
    /* Policy:  Don't attempt a sync update if registration for async
       updates have not succeeded.. */
    if (mdi.mgmt_info.receive_state == HASIM_MGMT_RECEIVE_REGISTERED)
      {
      if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
        {
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
            if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
              {
              mdi.mgmt_info.last_sync_check = 0;

              info_get_fail = 0;

              /* Get MGMT base information.. */
              ps.session.type = PMS_SR_CALL_RETURN_TIMED;
              ps.session.info.crt.time = 0;
              ps.payload.type = PMS_PD_MGMT_INFO_GET_EXECUTE;

              if (pms_send(&ps, &pr) == 0)
                {
                if (pr.payload.data.mgmt_info_get_status.err == \
                  PMS_PD_MGMT_INFO_GET_STATUS_SUCCESS)
                  {
                  mgmt_state = pr.payload.data.mgmt_info_get_status.state;
                  }
                else
                  {
                  info_get_fail = 1;
                  }
                }
              else
                {
                info_get_fail = 1;
                }


              /* Get MGMT rnd address information.. */

              ps.session.type = PMS_SR_CALL_RETURN_TIMED;
              ps.session.info.crt.time = 0;
              ps.payload.type = PMS_PD_MGMT_RND_ADDRESS_INFO_GET_EXECUTE;

              for(i=0;i<16;i++)
                {
                ps.payload.data.mgmt_rnd_address_info_get_execute.index = i;

                if (pms_send(&ps, &pr) == 0)
                  {
                  if (pr.payload.data.mgmt_rnd_address_info_get_status.err == \
                    PMS_PD_MGMT_RND_ADDRESS_INFO_GET_STATUS_ERR_NONE)
                    {
                    rnd_address_identifier[i] = \
                    pr.payload.data.mgmt_rnd_address_info_get_status.identifier;
                    strncpy(&rnd_address_node_ip_address[i][0], \
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
&pr.payload.data.mgmt_rnd_address_info_get_status.node_ip_address[0], 20);
                 strncpy(&rnd_address_drawer_ip_address[i][0], \
&pr.payload.data.mgmt_rnd_address_info_get_status.drawer_ip_address[0], 20);
                 rnd_address_node_slot_number[i] = \
pr.payload.data.mgmt_rnd_address_info_get_status.node_slot_number;
                 }
               else
                 {
                 info_get_fail = 1;
                 }
               }
             else
               {
               info_get_fail = 1;
               }
             }

           /* Only mark MGMT update as successful if all pieces of data
              were received successfully.. */

           if (info_get_fail == 0)
             {
             mdi.mgmt_info.mgmt_state_cache = mgmt_state;

             for(i=0;i<16;i++)
               {
               mdi.mgmt_info.rnd_address_cache[i].identifier = \
                 rnd_address_identifier[i];
            strncpy(&mdi.mgmt_info.rnd_address_cache[i].node_ip_address[0], \
                 &rnd_address_node_ip_address[i][0], 20);
strncpy(&mdi.mgmt_info.rnd_address_cache[i].drawer_ip_address[0],\
                 &rnd_address_drawer_ip_address[i][0], 20);
               mdi.mgmt_info.rnd_address_cache[i].node_slot_number = \
                 rnd_address_node_slot_number[i];
               }

             mdi.mgmt_info.last_update = 0;
             }

           }
         }
       }
     }
   else
     {
     mdi.mgmt_info.last_sync_check = 0;
     }
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
    }


  /* Validity Check */

  /* Process cache state validity transitions.  The policy is on a MGMT cache
     transition to invalid, return state variables to initial configuration.. */

  if(mdi.mgmt_info.last_update < HASIM_CHECK_VALID_INTERVAL)
    {
    if (mdi.mgmt_info.cache_state != HASIM_MGMT_CACHE_VALID)
      mdi.mgmt_info.cache_state = HASIM_MGMT_CACHE_VALID;
    }
  else if((mdi.mgmt_info.last_update >= HASIM_CHECK_VALID_INTERVAL && \
    mdi.mgmt_info.last_update < HASIM_CHECK_INVALID_INTERVAL))
    {
    if (mdi.mgmt_info.cache_state == HASIM_MGMT_CACHE_VALID)
      mdi.mgmt_info.cache_state = HASIM_MGMT_CACHE_OLD;
    }
  else if(mdi.mgmt_info.last_update >= HASIM_CHECK_INVALID_INTERVAL)
    {
    if (mdi.mgmt_info.cache_state == HASIM_MGMT_CACHE_OLD)
      {

      /* RND */

      for(i=0;i<16;i++)
        {

        mdi.rnd_info[i].cache_state = HASIM_RND_CACHE_INVALID;
        mdi.rnd_info[i].last_update = HASIM_CHECK_INVALID_INTERVAL;

       if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED) != 0)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_RND_STATUS;
          pr.payload.data.rnd_status.identifier = \
            mdi.mgmt_info.rnd_address_identifier[i];
          pms_receive(&pr, 0, 0);
          mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
          }

      if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
              != 0)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_RND_MD0_STATUS;
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
           }

         }


       /* NODE*/

       mdi.node_info.cache_state = HASIM_NODE_CACHE_INVALID;
       mdi.node_info.last_update = HASIM_CHECK_INVALID_INTERVAL;

       if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
         {
         mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
         }

       if (mdi.node_info.rg0_app_name_state ==\
             HASIM_NODE_RG0_APP_NAME_REGISTERED)
         {
        mdi.node_info.rg0_app_name_state = HASIM_NODE_RG0_APP_NAME_UNREGISTERED;
         }

       if ((mdi.node_info.receive_state &\
              HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED) != 0)
         {
         pr.session.type = PMS_SR_CALL_NO_RETURN;
         pr.payload.type = PMS_PD_NODE_RG0_STATUS;
         pms_receive(&pr, 0, 0);
         mdi.node_info.receive_state &=\
           !HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED;
         }

       if ((mdi.node_info.receive_state & \
         HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED) != 0)
         {
         pr.session.type = PMS_SR_CALL_RETURN_TIMED;
         pr.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE;
         pms_receive(&pr, 0, 0);
         mdi.node_info.receive_state &=\
          !HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED;
         }


       /* MGMT */
```

CODE EXAMPLE 7-5 PMS Client User and Management Interface *(Continued)*

```
        mdi.mgmt_info.cache_state = HASIM_MGMT_CACHE_INVALID;

        for(i=0;i<16;i++)
          mdi.mgmt_info.rnd_address_identifier[i] = -1;

        if (mdi.mgmt_info.rnd_address_state == HASIM_MGMT_RND_ADDRESS_VERIFIED)
          {
          mdi.mgmt_info.rnd_address_state = HASIM_MGMT_RND_ADDRESS_UNVERIFIED;
          }

        if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE)
          {
          mdi.mgmt_info.pms_state = HASIM_MGMT_PMS_STATE_UNAVAILABLE;
          }

        if (mdi.mgmt_info.receive_state == HASIM_MGMT_RECEIVE_REGISTERED)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_MGMT_STATUS;
          pms_receive(&pr, 0, 1);

          mdi.mgmt_info.receive_state = HASIM_MGMT_RECEIVE_UNREGISTERED;
          }


        /* USER */

        if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_USER_STATUS;
          pms_receive(&pr, 0, 0);

          mdi.user_info.receive_state = HASIM_USER_RECEIVE_UNREGISTERED;
          }

        if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
          {
          pms_disconnect();

          mdi.user_info.pms_view = HASIM_USER_PMS_VIEW_UNREACHABLE;
          }
        }
      }
}
```

The following example shows the PMS client node interface.

CODE EXAMPLE 7-6 PMS Client Node Interface

```
void
app_hasim_node_process(void)
{

  struct pms_send       ps;
  struct pms_receive    pr;

  int                   info_get_fail;

  int                   rg0_state;

  int                   i;


  /* Receive Check */

  /* If NODE messages are not receive registered, attempt to register them if PMS
     is in the available state and reachable, and if USER receive messages are
     registered.  If registration is successful, force an initial cache
     update.. */

  if (mdi.node_info.receive_state != HASIM_NODE_GROUP_RECEIVE_REGISTERED)
    {
    if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE)
      {
      if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
        {
        if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
          {

          if ((mdi.node_info.receive_state & \
             HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED) == 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_NODE_RG0_STATUS;
            if (pms_receive(&pr, app_hasim_receive_post, 0) != -1)
              mdi.node_info.receive_state |= \
                 HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED;
            }

          if ((mdi.node_info.receive_state & \
            HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED) == 0)
            {
            pr.session.type = PMS_SR_CALL_RETURN_TIMED;
            pr.session.info.crt.time = 50;
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
          pr.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE;
          if (pms_receive(&pr, app_hasim_receive_post, 0) != -1)
            mdi.node_info.receive_state |= \
               HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED;
          }

        /* Force an info_get immediately after registering.. */
        mdi.node_info.last_sync_check = HASIM_SYNCCHECK_INTERVAL;
        }
      }
    }
  }


/* Name Check */

/* If this application's name is not registered, register it if PMS is
   available and reachable, and if USER registration is complete.. */

if (mdi.node_info.rg0_app_name_state != HASIM_NODE_RG0_APP_NAME_REGISTERED)
  {
  if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE)
    {
    if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
      {
      if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
        {

        /* Set NODE RG0 application name.. */

        ps.session.type = PMS_SR_CALL_RETURN_TIMED;
        ps.session.info.crt.time = 0;
        ps.payload.type = PMS_PD_NODE_RG0_APP_NAME_EXECUTE;
        strcpy(&ps.payload.data.node_rg0_app_name_execute.name[0], \
          "hasim");
        ps.payload.data.node_rg0_app_name_execute.command = \
          PMS_PD_NODE_RG0_APP_NAME_EXECUTE_ADD;

        if (pms_send(&ps, &pr) == 0)
          {
          if (pr.payload.data.node_rg0_app_name_status.err == \
            PMS_PD_NODE_RG0_APP_NAME_STATUS_ERR_NONE)
            {
            mdi.node_info.rg0_app_name_state = \
               HASIM_NODE_RG0_APP_NAME_REGISTERED;
            }
          }
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
          }
        }
      }
    }


  /* Service State check */

  /* Process application service state transitions.  On an active-to-offline
     transition, return state variables to a pre-RND configuration.  This
     example's applications policy does not monitor RND pairs
     if it is offline.. */

  if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_OFFLINE)
    {
    if (mdi.node_info.cache_state != HASIM_NODE_CACHE_INVALID)
      {
      if (mdi.node_info.rg0_state_cache != \
          PMS_PD_NODE_RG0_INFO_GET_STATUS_OFFLINE)
        {
        mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_ACTIVE;
        }
      }
    }
  else /* mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE */
    {
    if (mdi.node_info.cache_state != HASIM_NODE_CACHE_INVALID)
      {
      if (mdi.node_info.rg0_state_cache == \
          PMS_PD_NODE_RG0_INFO_GET_STATUS_OFFLINE)
        {

        /* RND */

        for(i=0;i<16;i++)
          {

          if ((mdi.rnd_info[i].receive_state & \
               HASIM_RND_RECEIVE_REGISTERED) != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
             }

         if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
              != 0)
             {
             pr.session.type = PMS_SR_CALL_NO_RETURN;
             pr.payload.type = PMS_PD_RND_MD0_STATUS;
             pr.payload.data.rnd_status.identifier = \
               mdi.mgmt_info.rnd_address_identifier[i];
             pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
             }

           }


         mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
         }
       }
     }


   /* Sync Check */

   /* Policy: Sync update checked every SYNCCHECK_INTERVAL seconds.. */
   if (mdi.node_info.last_sync_check > HASIM_SYNCCHECK_INTERVAL)
     {
     /* Policy:  Don't attempt a sync update if any async partial updates have
        been received within SYNCCHECK_INTERVAL.. */
     if (mdi.node_info.last_update > HASIM_SYNCCHECK_INTERVAL)
       {
       /* Policy:  Don't attempt a sync update if registration for async
          updates have not succeeded.. */
       if (mdi.node_info.receive_state == HASIM_NODE_GROUP_RECEIVE_REGISTERED)
         {
         if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
           {
           if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
             {
             mdi.node_info.last_sync_check = 0;

             info_get_fail = 0;

             /* Get NODE RG0 information.. */
             ps.session.type = PMS_SR_CALL_RETURN_TIMED;
             ps.session.info.crt.time = 0;
             ps.payload.type = PMS_PD_NODE_RG0_INFO_GET_EXECUTE;
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
            if (pms_send(&ps, &pr) == 0)
              {
              if (pr.payload.data.node_rg0_info_get_status.err == \
                PMS_PD_NODE_RG0_INFO_GET_STATUS_SUCCESS)
                {
                rg0_state = pr.payload.data.node_rg0_info_get_status.state;
                }
              else
                {
                info_get_fail = 1;
                }
              }
            else
              {
              info_get_fail = 1;
              }


            /* Get any other NODE info? */


           /* Only mark NODE update as successful if all pieces of data gotten
               were received successfully.. */

            if (info_get_fail == 0)
              {
              mdi.node_info.rg0_state_cache = rg0_state;

              mdi.node_info.last_update = 0;
              }

            }
          }
        }
      }
    else
      {
      mdi.node_info.last_sync_check = 0;
      }
    }


  /* Validity Check */

 /* Process cache state validity transitions.  The policy is that on a NODE cache
     transition to invalid, NODE AND RND state variables are returned to an
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
      initial configuration.. */

   if(mdi.node_info.last_update < HASIM_CHECK_VALID_INTERVAL)
     {
     if (mdi.node_info.cache_state != HASIM_NODE_CACHE_VALID)
       mdi.node_info.cache_state = HASIM_NODE_CACHE_VALID;
     }
   else if((mdi.node_info.last_update >= HASIM_CHECK_VALID_INTERVAL && \
     mdi.node_info.last_update < HASIM_CHECK_INVALID_INTERVAL))
     {
     if (mdi.node_info.cache_state == HASIM_NODE_CACHE_VALID)
       mdi.node_info.cache_state = HASIM_NODE_CACHE_OLD;
     }
   else if(mdi.node_info.last_update >= HASIM_CHECK_INVALID_INTERVAL)
     {
     if (mdi.node_info.cache_state == HASIM_NODE_CACHE_OLD)
       {

       /* RND */

       for(i=0;i<16;i++)
         {

        if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED) != 0)
           {
           pr.session.type = PMS_SR_CALL_NO_RETURN;
           pr.payload.type = PMS_PD_RND_STATUS;
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
           }

        if ((mdi.rnd_info[i].receive_state & HASIM_RND_MD0_RECEIVE_REGISTERED)\
             != 0)
           {
           pr.session.type = PMS_SR_CALL_NO_RETURN;
           pr.payload.type = PMS_PD_RND_MD0_STATUS;
           pr.payload.data.rnd_status.identifier = \
             mdi.mgmt_info.rnd_address_identifier[i];
           pms_receive(&pr, 0, 0);
           mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
           }

         }
```

CODE EXAMPLE 7-6 PMS Client Node Interface *(Continued)*

```
        /* NODE*/

        if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
          {
          mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
          }

        if (mdi.node_info.rg0_app_name_state == \
            HASIM_NODE_RG0_APP_NAME_REGISTERED)
          {
        mdi.node_info.rg0_app_name_state = HASIM_NODE_RG0_APP_NAME_UNREGISTERED;
          }

        if ((mdi.node_info.receive_state & \
            HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED) != 0)
          {
          pr.session.type = PMS_SR_CALL_NO_RETURN;
          pr.payload.type = PMS_PD_NODE_RG0_STATUS;
          pms_receive(&pr, 0, 0);
          mdi.node_info.receive_state &= \
           !HASIM_NODE_RG0_STATUS_RECEIVE_REGISTERED;
          }

        if ((mdi.node_info.receive_state & \
            HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED) != 0)
          {
          pr.session.type = PMS_SR_CALL_RETURN_TIMED;
          pr.payload.type = PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE;
          pms_receive(&pr, 0, 0);
          mdi.node_info.receive_state &= \
            !HASIM_NODE_RG0_APP_STATE_SET_EXECUTE_RECEIVE_REGISTERED;
          }

        }
      }


 }
```

The following example shows a PMS client RND interface.

CODE EXAMPLE 7-7 PMS Client RND Interface

```
void
app_hasim_rnd_process(void)
{

  struct pms_send       ps;
  struct pms_receive    pr;

  int                   info_get_fail;

  int                   view;
  int                   md0_config;

  int                   i;


  /* Receive Check */

  /* If RND messages are not receive registered, attempt to register for
     in-use RND address list entries if the service state is active, if
     PMS is in the available state and reachable, and if USER receive messages
     are registered.  If registration is successful, force an initial cache
     update.. */

  for(i=0;i<16;i++)
    {
    if (mdi.rnd_info[i].receive_state != HASIM_RND_GROUP_RECEIVE_REGISTERED)
      {
      if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
        {
        if (mdi.mgmt_info.rnd_address_identifier[i] != -1)
          {
          if (mdi.mgmt_info.pms_state == HASIM_MGMT_PMS_STATE_AVAILABLE)
            {
            if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
              {
              if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
                {

                if ((mdi.rnd_info[i].receive_state & \
                     HASIM_RND_RECEIVE_REGISTERED) == 0)
                  {
                  pr.session.type = PMS_SR_CALL_NO_RETURN;
                  pr.payload.type = PMS_PD_RND_STATUS;
                  pr.payload.data.rnd_status.identifier = \
```

CODE EXAMPLE 7-7 PMS Client RND Interface *(Continued)*

```
                    mdi.mgmt_info.rnd_address_cache[i].identifier;
                  if (pms_receive(&pr, app_hasim_receive_post, 0) != -1)
                 mdi.rnd_info[i].receive_state |= HASIM_RND_RECEIVE_REGISTERED;
                  }

                if ((mdi.rnd_info[i].receive_state & \
                     HASIM_RND_MD0_RECEIVE_REGISTERED) == 0)
                  {
                  pr.session.type = PMS_SR_CALL_NO_RETURN;
                  pr.payload.type = PMS_PD_RND_MD0_STATUS;
                  pr.payload.data.rnd_md0_status.identifier = \
                    mdi.mgmt_info.rnd_address_cache[i].identifier;
                  if (pms_receive(&pr, app_hasim_receive_post, 0) != -1)
                    mdi.rnd_info[i].receive_state |= \
                     HASIM_RND_MD0_RECEIVE_REGISTERED;
                  }

                /* Force an info_get immediately after registering.. */
                mdi.rnd_info[i].last_sync_check = HASIM_SYNCCHECK_INTERVAL;
                }
              }
            }
          }
        }
      }
    }



  /* Sync Check */

  for(i=0;i<16;i++)
    {
    /* Policy:  Sync update checked every SYNCCHECK_INTERVAL seconds.. */
    if (mdi.rnd_info[i].last_sync_check > HASIM_SYNCCHECK_INTERVAL)
      {
      /* Policy:  Don't attempt a sync update if any async partial updates have
         been received within SYNCCHECK_INTERVAL.. */
      if (mdi.rnd_info[i].last_update > HASIM_SYNCCHECK_INTERVAL)
        {
        /* Policy:  Don't attempt a sync update if registration for async
           updates have not succeeded.. */
       if (mdi.rnd_info[i].receive_state == HASIM_RND_GROUP_RECEIVE_REGISTERED)
          {
          if (mdi.user_info.receive_state == HASIM_USER_RECEIVE_REGISTERED)
            {
            if (mdi.user_info.pms_view == HASIM_USER_PMS_VIEW_REACHABLE)
```

CODE EXAMPLE 7-7 PMS Client RND Interface *(Continued)*

```
            {
            mdi.rnd_info[i].last_sync_check = 0;

            info_get_fail = 0;

            /* Get RND information.. */
            ps.session.type = PMS_SR_CALL_RETURN_TIMED;
            ps.session.info.crt.time = 0;
            ps.payload.type = PMS_PD_RND_INFO_GET_EXECUTE;
            ps.payload.data.rnd_info_get_execute.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];

            if (pms_send(&ps, &pr) == 0)
              {
              if (pr.payload.data.rnd_info_get_status.err == \
                PMS_PD_RND_INFO_GET_STATUS_ERR_NONE)
                {
                view = pr.payload.data.rnd_info_get_status.view;
                }
              else
                {
                info_get_fail = 1;
                }
              }
            else
              {
              info_get_fail = 1;
              }

            /* Get RND MD0 information.. */
            ps.session.type = PMS_SR_CALL_RETURN_TIMED;
            ps.session.info.crt.time = 0;
            ps.payload.type = PMS_PD_RND_MD0_INFO_GET_EXECUTE;
            ps.payload.data.rnd_md0_info_get_execute.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];

            if (pms_send(&ps, &pr) == 0)
              {
              if (pr.payload.data.rnd_md0_info_get_status.err == \
                PMS_PD_RND_MD0_INFO_GET_STATUS_ERR_NONE)
                {
                md0_config = pr.payload.data.rnd_md0_info_get_status.config;
                }
              else
                {
                info_get_fail = 1;
                }
```

CODE EXAMPLE 7-7 PMS Client RND Interface *(Continued)*

```
                }
              else
                {
                info_get_fail = 1;
                }


              /* Only mark MGMT update as successful if all pieces of data
                 were received successfully.. */

              if (info_get_fail == 0)
                {
                mdi.rnd_info[i].view_cache = view;

                mdi.rnd_info[i].md0_config_cache = md0_config;

                mdi.rnd_info[i].last_update = 0;
                }

              }
            }
          }
        }
      else
        {
        mdi.rnd_info[i].last_sync_check = 0;
        }
      }
    }


  /* Validity Check */

  /* Process cache state validity transitions.  The policy is on a RND cache
    transition to invalid, return RND state variables for the pair to an initial
     configuration.. */

  for(i=0;i<16;i++)
    {
    if(mdi.rnd_info[i].last_update < HASIM_CHECK_VALID_INTERVAL)
      {
      if (mdi.rnd_info[i].cache_state != HASIM_RND_CACHE_VALID)
        mdi.rnd_info[i].cache_state = HASIM_RND_CACHE_VALID;
      }
    else if((mdi.rnd_info[i].last_update >= HASIM_CHECK_VALID_INTERVAL && \
      mdi.rnd_info[i].last_update < HASIM_CHECK_INVALID_INTERVAL))
      {
```

CODE EXAMPLE 7-7 PMS Client RND Interface *(Continued)*

```
        if (mdi.rnd_info[i].cache_state == HASIM_RND_CACHE_VALID)
          mdi.rnd_info[i].cache_state = HASIM_RND_CACHE_OLD;
        }
      else if(mdi.rnd_info[i].last_update >= HASIM_CHECK_INVALID_INTERVAL)
        {
        if (mdi.rnd_info[i].cache_state == HASIM_RND_CACHE_OLD)
          {

          /* RND */

         if ((mdi.rnd_info[i].receive_state & HASIM_RND_RECEIVE_REGISTERED) != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_RECEIVE_REGISTERED;
            }

          if ((mdi.rnd_info[i].receive_state & \
              HASIM_RND_MD0_RECEIVE_REGISTERED) != 0)
            {
            pr.session.type = PMS_SR_CALL_NO_RETURN;
            pr.payload.type = PMS_PD_RND_MD0_STATUS;
            pr.payload.data.rnd_status.identifier = \
              mdi.mgmt_info.rnd_address_identifier[i];
            pms_receive(&pr, 0, 0);
            mdi.rnd_info[i].receive_state &= !HASIM_RND_MD0_RECEIVE_REGISTERED;
            }

          }
        }
      }
}
```

CHAPTER **8**

Solaris Operating System APIs

This chapter introduces Solaris operating system APIs of concern to the Netra CT server, including configuration and status of the system frutree and environmental monitoring with sensor status information. This is handled through the Platform Information and Control Library (PICL) framework, gathering FRU-ID information, and dynamic reconfiguration interfaces. These subjects are addressed in:

Solaris Operating System PICL Framework

PICL provides a method to publish platform-specific information for clients to access in a way that is not specific to the platform. The Solaris PICL framework provides information about the system configuration which it maintains in the PICL tree. Within this PICL tree is a subtree named *frutree*, that represents the hierarchy of system FRUs with respect to a root node in the tree called *chassis*. The frutree represents physical resources of the system.

The main components of the PICL framework are:

- PICL interface (`libpicl.so`) – Implements the generic platform-independent interface that clients can use to access the platform-specific information.

- PICL tree (`libpicltree.so`) – A repository of all the nodes and properties representing the platform configuration.
- PICL plug-in modules – Shared objects that publish platform-specific data in the PICL tree.
- PICL daemon (`picld`) – Maintains and controls access to the PICL information from clients and from PICL plug-in modules.

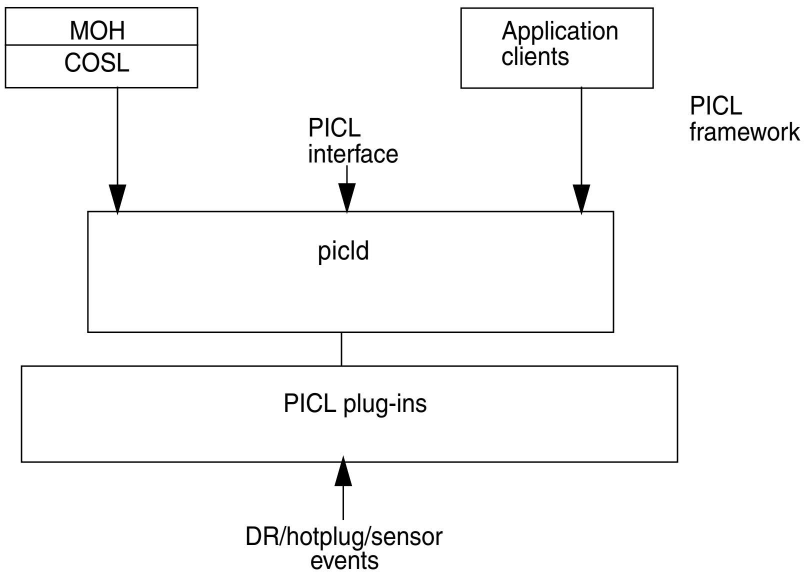


FIGURE 8-1 PICL Daemon (`picld`) and Plug-ins

FIGURE 8-1 diagrams the PICL daemon (`picld`) and its connection to the PICL plug-ins, some of which are common to all platforms. These plug-ins are responsible for populating and maintaining the PICL tree during system boot and dynamic reconfiguration (DR) operations. They also handle sensor events.

Application clients use `libpicl`(3LIB) to interface with picld to display or set parameters using Managed Object Hierarchy (MOH), RMI, SNMP, or Solaris PICL client interfaces. MOH uses the common operating system library (COSL) interface to access `picld`, which in turn uses the `libpicl` interfaces.

Updates to the system frutree are done during DR operations, which are performed using `cfgadm` (1M), or during fru latch and unlatch operations.

The following section identifies the exported interfaces in the PICL tree which are basically the nodes and the properties of the node that are present in the PICL tree.

PICL Frutree Topology

To read the PICL frutree data of the system, use the `prtpicl`(1M) command. The structure of the PICL frutree involves a hierarchical representation of nodes. The immediate descendants of /frutree are one or more fru nodes by the name of **chassis**.

FRUs and their locations are represented by nodes of classes **fru** and **location** respectively, in the PICL frutree under the /frutree node. The **port** node is an extension of the fru class.

The three major node classes, location, fru, and port, are summarized in TABLE 8-1. Each of these classes is populated with various properties, among which are State and Condition. More detailed information is provided in the sections following this summary table.

TABLE 8-1 PICL FRUtree Topology Summary

Node Class	Properties	Description
location	SlotType	Type of location.
	Label	Slot Label information.
	GeoAddr	Geographical address.
	StatusTime	Time when State was updated last.
	Bus-addr	Bus address.
	State	State of the location: empty, connected, disconnected, or unknown.
fru	FruType	Type of FRU.
	Devices	Table of node handles in platform tree.
	State	State of the FRU: configured, unconfigured, or unknown.
	StatusTime	Time when State was updated last.
	Condition	Condition or operational state of the FRU: ok, failing, failed, unknown, or unusable.
	ConditionTime	Time when Condition was updated last.

TABLE 8-1 PICL FRUtree Topology Summary

Node Class	Properties	Description
port	Bus-addr	Bus address of port: network, serial, or parallel.
	GeoAddr	Geographical address of port.
	Label	Label information.
	PortType	Type of port.
	State	State of the port: up, down, or unknown.
	StatusTime	Time when State was updated last.
	Condition	Condition of the port: ok, unknown, failing, or failed.
	ConditionTime	Time when Condition was updated last.
	Devices	Table of node handles in platform device tree.

Chassis Node Property Updates

In addition to those properties already defined by PICL, the following property is added:

ChassisType

`CHARSTRING` read-only

The **ChassisType** read-only property represents the chassis type. The value of ChassisType is currently defined as:

```
uname -i
```

There should be a configuration file of this name with the `.conf` extension in the `/usr/platform/'uname -i'/lib/picl/plugins/` directory. If none is provided, then the frutree is not initialized.

Fru Class Properties

Where the following **fru** class properties are writeable, permission checks govern that they be written to by a process with the user ID of root.

Fru State

CHARSTRING read-only

The **State** property of the fru class node represents the **occupant** state of the cfgadm attachment point associated with fru node. In such a case, a read operation of this property directs the plug-in to check the state of the occupant using libcfgadm to determine the latest State information.

The various state values are shown in TABLE 8-2.

TABLE 8-2 PICL FRU State Value Properties

State	Property Value Description
unconfigured	The FRU is not configured and unusable. See cfgadm(1M) for details.
configured	The FRU is configured and usable. See cfgadm(1M) for details.

Fru Condition

CHARSTRING read-only

The **Condition** property of the fru class node represents the condition of occupant of the cfgadm attachment point. The various condition values are shown in TABLE 8-3. When libcfgadm interfaces are not available, a platform must provide the same semantics using platform-specific interfaces in defining this property.

TABLE 8-3 PICL FRU Condition Value Properties

Condition	Property Value Description
unknown	FRU condition could not be determined. See cfgadm(1M) for details.
ok	FRU is functioning as expected. See cfgadm(1M) for details.
failing	A recoverable fault was found. See cfgadm(1M) for details.
failed	An unrecoverable fault was found. See cfgadm(1M) for details.
unusable	FRU is unusable for undetermined reason. See cfgadm(1M) for details.

Either the FRU software such as drivers or applications, or FRU hardware, can be responsible for providing the Condition information. This information might be either polled for, or retrieved asynchronously via an event mechanism such as sysevent.

Port Class Node

The connectivity between nodes in a telco network is established by a link that provides the physical transmission medium. A **port** node represents a resource of a fru that provides such a link. Examples of ports are: serial port and network port.

The port class node extends the PICL frutree definition of fru class of nodes. A port is always a child of a fru class, even if it is the only resource of the fru. There are no location or fru nodes beneath a port class node, because FRUs linked to the port class node are not managed in the domain in which port class node exists. There might be dependencies, such as when a remote device is cabled to a port node. These dependencies can influence the state of the port, but not necessarily the FRU itself.

The PICL frutree plug-in is responsible for identifying the port class nodes and creating the respective nodes in the frutree.

Note – The port class node should not be associated with USB port or SCSI port. These are locations into which a FRU can be plugged, become visible to the system CPU, and managed by it. FRUs beyond the port class of nodes are not visible to the CPU.

Port Class Properties

Port class properties consist of **State** and **Condition**, values of which are shown in the following paragraphs.

State

`CHARSTRING` read-only

A port class node can be in one of the states shown in TABLE 8-4:

TABLE 8-4 Port Class State Values

Port State Values	**Description**
Down	A port is down when its link state is down, that is, a carrier was not detected.
Up	A port is up when its link state is up, that is, a carrier is detected.
Unknown	The plug-in cannot determine the state of the port.

The state of the port node is maintained by the frutree plug-in. The **State** value is initially determined by looking at the `kstat` information published by the device driver that owns the port. If the device driver information is not determined, this value remains unknown. The parent fru of the port must set its state to `configured` for the port to be anything other than `unknown`. See `kstat`(1M) for details.

Condition

`CHARSTRING` read-write

The **Condition** value of a port class node carries the same meaning as the `cfgadm` value of the attachment point, as shown in TABLE 8-5.

TABLE 8-5 Port Condition Values

Port Condition Values	**Description**
`ok`	Port is functioning as expected.
`failing`	A predictive failure has been detected. This typically occurs when the number of correctable errors exceeds a threshold.
`failed`	Port has failed. It cannot transmit or receive data due to an internal fault. This indicates a broken path within the FRU, and not external to the FRU which would be denoted by its link state.
`unknown`	Port condition could not be determined.

Initial Condition values can be obtained by looking at the driver `kstat` information, if present. A device driver managing a resource of the FRU can influence the overall condition of the FRU by sending appropriate fault events. The property information is valid only when the parent fru state is configured.

PortType

`CHARSTRING` read-only

This PortType property indicates the functional class of port device, as shown in TABLE 8-6.

TABLE 8-6 PortType Property Values

PortType Values	**Description**
`network`	Represents a network device.
`serial`	Represents a serial device.
`parallel`	Represents a parallel port.

Common Property Updates

The following properties are common to all PICL classes:

GeoAddr

`UINT` read-only

This property indicates the geographical address of the node in relation to its parent node. It should be possible to point to the physical location (slot number) of the node in its parent domain. For example, the Netra CT 810 server describes a location's GeoAddr under the chassis node as its physical slot number. This could differ from the Label information printed on the chassis itself. In this instance, the system controller slot on the Netra CT 810 system chassis is labelled as CPU, although its GeoAddr has a value of 1. Note that the Label property might not have the physical slot number embedded in it.

StatusTime

`TIMESTAMP` read-only

This property indicates when the State property was last updated. This can indicate when a FRU was last inserted or removed, configured or unconfigured, or when the port link went down. Status time is updated even for transitional state changes.

ConditionTime

`TIMESTAMP` read-only

This property indicates when the Condition property was last updated. Using this property, for example, a System Management software can calculate how long a fru/port has been in operation before failure.

Temperature Sensor Node State

CHARSTRING

A temperature sensor node is in the PICL frutree under the Environment property of the fru node.

The temperature sensors are represented as `PICL_CLASS_TEMPERATURE_SENSOR` class in the PICL tree. A State property is declared for each temperature sensor node representing the state information as shown in TABLE 8-7.

TABLE 8-7 State Property Values for Temperature Sensor Node

State Property Values	Description
ok	Environment state is OK.
warning	Environment state is warning, (that is, current temperature is below/above lower/upper warning temperature).
failed	Environment state is failed (that is, current temperature is below/above lower/upper critical temperature).
unknown	Environment state is unknown (that is, current temperature cannot be determined).

PICL Man Page References

TABLE 8-8 lists the Solaris OS man pages that document the PICL framework and API. You can view the following man pages at the command line or on the Solaris OS documentation web site (`http://docs.sun.com/documentation`).

TABLE 8-8 PICL Man Pages

Man Page	Description
`picld`(1M)	Describes how the daemon initializes plug-in modules at startup. The man page also describes the PICL tree and PICL plug-in modules.
`libpicl`(3LIB)	Lists the library functions clients use to interface with the PICL daemon in order to access information from the PICL tree.
`libpicl`(3PICL)	Client API for sending requests to the PICL daemon to access the PICL tree.

TABLE 8-8 PICL Man Pages *(Continued)*

Man Page	**Description**
picld_log(3PICLTREE)	Describes the function the PICL daemon and the plug-in modules use to log messages and inform users of any error or warning conditions.
picl_plugin_register(3PICLTREE)	Describes the function plug-in modules use to register itself with the PICL daemon.
prtpicl(1M)	Prints the PICL tree. The prtpicl command prints the PICL tree maintained by the PICL daemon. The output of prtpicl includes the name and PICL class of the nodes.
(3LIB) **Functions**	
picl_initialize(3PICL)	Initiates a session with the PICL daemon.
picl_get_first_prop(3PICL)	Gets a property handle of a node.
picl_get_next_by_col(3PICL)	Accesses a table property.
picl_get_next_by_row(3PICL)	Accesses a table property.
picl_get_next_prop(3PICL)	Gets a property handle of a node.
picl_get_prop_by_name(3PICL)	Gets the handle of the property by name.
picl_get_propinfo(3PICL)	Gets the information about a property.
picl_get_propinfo_by_name(3PICL)	Gets property information and handle of a property by name.
picl_get_propval(3PICL)	Gets the value of a property.
picl_get_propval_by_name(3PICL)	Gets the value of a property by name.
picl_get_root(3PICL)	Gets the root handle of the PICL tree.
picl_set_propval(3PICL)	Sets the value of a property to the specified value.
picl_set_propval_by_name(3PICL)	Sets the value of a named property to the specified value.
picl_shutdown(3PICL)	Shuts down the session with the PICL daemon.
picl_strerror(3PICL)	Gets error message string.
picl_wait(3PICL)	Waits for PICL tree to refresh.
picl_walk_tree_by_class(3PICL)	Walks subtree by class.

For examples of use of these functions, see "Programming Watchdog Timers Using the PICL API" on page 283.

Dynamic Reconfiguration Interfaces

The Dynamic Reconfiguration (DR) interfaces allow resources to be reconfigured without user intervention when system resources are added or removed while the system is running. Traditionally, applications assume that OS resources remain static after boot. In DR situations, challenges faced by applications include the following:

- Addition or availability of new devices. Applications might want to be notified in order to make use of the newly added resources.
- Removal of devices. Applications need to be notified of pending resource removal from the system so they can either block or prepare for the pending operation.

The Solaris OS has knowledge of DR operations, but certain applications might not. If an application is holding the resources involved in the DR operation, the operation will fail. To be successful, applications need to be dynamically aware of the current state of the system. The Solaris DR framework includes the Reconfiguration Coordination Manager (RCM), `cfgadm`(1m), and `libcfgadm` (3LIB). It also includes the PCI hotplug/cPCI Hotswap framework (`cfgadm_pci`(1M)), SCSI hotplug framework (`cfgadm_scsi`(1M)), and the Hotswap Controller driver (cphsc(7D)).

The following sections describe:

- "Reconfiguration Coordination Manager" on page 279
- "Hot-Swap Support" on page 279
- "Configuration Administration (cfgadm)" on page 280

Reconfiguration Coordination Manager

The Reconfiguration Coordination Manager (RCM) is a generic framework which allows DR to interact with system management software. The framework enables automated DR removal operations on platforms with proper software and hardware configuration. RCM defines generic APIs to coordinate DR operations between DR initiators and DR clients during resource removal. For details on RCM, go to `http://www.sun.com/documentation`.

Hot-Swap Support

The Netra CT server supports the following three hot-swap models according to the PICMG CompactPCI Hotswap specifications version 2.1 R1.0:

- Basic hot swap
- Full hot swap
- High availability hot swap

These models can be described by two terms:

- Hardware connection process — the electrical connection (and disconnection) of an I/O board
- Software connection process — the software configuration (and unconfiguration) of the I/O board by the operating system (allocating/releasing PCI resources, attaching/detaching device drivers, and so on.)

In the basic hot-swap model, the hardware connection process can be performed automatically by the hardware, while the software connection process requires operator assistance.

In the full hot-swap model, both the hardware and the software connection processes are performed automatically. The Netra CT server is configured for full hot swap by default. The mode of a slot can be reconfigured to basic hot swap using the `cfgadm` command in cases where a third-party board does not support full hot swap.

In the high-availability model, software has the capability of controlling the power-on of the FRU hardware, beyond the hardware and software connection processes. Drivers and services can isolate a board from the system until an operator is able to intervene.

The Netra CT server uses the `cfgadm`(1M) utility for administering the hot-swap process. This includes connecting and disconnecting, configuring and unconfiguring the hardware and software, and setting various operation modes. Elements of the b

utility are described in the next section.

On the Netra CT server, CPU card, CPU transition card, and I/O board hot swapping is supported. It should be noted that non-hotswap friendly devices can be supported only in basic hot-swap mode. See the *Netra CT Server Service Manual* (816-2482) for list of hot-swappable FRUs.

Configuration changes are handled in a coherent way, because DR and the Frutree management framework are integrated in PICL. PICL frutree properties and `cfgadm` attachment point elements are mapped one-to-one, which creates data consistency. All DR operations are coordinated with a service processor.

Configuration Administration (cfgadm)

Configuration administration of a dynamically reconfigurable system is carried out through `cfgadm`(1M), which can display status, invoke configuration state changes, and invoke hardware specific functions. See the *Netra CT System Administration Guide* (816-2486) for more information on the `cfgadm` utility.

The `libcfgadm`(3LIB) command can be used to display a library of configuration interfaces.

Use cfgadm to perform a connect operation on a cPCI FRU, for example:

- To power on a FRU
- To check for HEALTHY#
- To bring a FRU out of PCI_RST#

```
# cfgadm -c connect operation
```

Use cfgadm to perform a disconnect operation on a cPCI FRU, for example:

- To notify applications (via RCM)
- To assert PCI_RST#
- To power OFF a FRU

```
# cfgadm -c disconnect operation
```

Programming Temperature Sensors Using the PICL API

Temperature sensor states can be read using the libpicl API. The properties that are supported in a PICL temperature sensor class node are listed in TABLE 8-9.

TABLE 8-9 PICL Temperature Sensor Class Node Properties

Property	Type	Description
LowWarningThreshold	INT	Low threshold for warning
LowShutdownThreshold	INT	Low threshold for shutdown
LowPowerOffThreshold	INT	Low threshold for power off
HighWarningThreshold	INT	High threshold for warning
HighShutdownThreshold	INT	High threshold for shutdown
HighPowerOffThreshold	INT	High threshold for power off

The PICL plug-in receives these sensor events and updates the State property based on the information extracted from the IPMI message. It then posts a PICL event.

The threshold levels of the PICL node class temperature-sensor are:

- Warning
- Shutdown
- PowerOff

TABLE 8-10 lists the PICL threshold levels and their MOH equivalents.

TABLE 8-10 PICL Threshold Levels and MOH Equivalents

PICL Threshold levels	MOH Equivalent
LowWarningThreshold	LowerThresholdNonCritical
LowShutdownThreshold	LowerThresholdCritical
LowPowerOffThreshold	LowerThresholdFatal
HighWarningThreshold	UpperThresholdNonCritical
HighShutdownThreshold	UpperThresholdCritical
HighPowerOffThreshold	UpperThresholdFatal

To obtain a reading of temperature sensor states, type the `prtpicl -v` command:

```
# prtpicl -c temperature-sensor -v
```

PICL output of the temperature sensors on a Netra CT system is shown in CODE EXAMPLE 8-1.

CODE EXAMPLE 8-1 Example Output of PICL Temperature Sensors

```
# prtpicl -c temperature-sensor -v
 CPU-sensor (temperature-sensor, 3700000634)
             :State          ok
             :HighWarningThreshold  60
             :HighShutdownThreshold          65
             :HighPowerOffThreshold          70
             :LowWarningThreshold   3
             :LowShutdownThreshold  0
             :LowPowerOffThreshold  0
             :Temperature            30
             :GeoAddr        0x2
             :Label          Ambient
             :_class         temperature-sensor
             :name           CPU-sensor
```

Note – PICL clients can use the `libpicl` APIs to set and get various properties of this sensor.

Programming Watchdog Timers Using the PICL API

The Netra CT system's watchdog service captures catastrophic faults in the Solaris OS running on either a host or satellite CPU board. The watchdog service reports such faults to the alarm card by means of either an IPMI message or by a de-assertion of the CPU's HEALTHY# signal.

The Netra CT system management controller provides two watchdog timers, the watchdog level 2 (WD2) timer and the watchdog level 1 (WD1) timer. Systems management software starts and the Solaris OS periodically pats the timers before they expire. If the WD2 timer expires, the watchdog function of the WD2 timer forces the SPARC™ processor to optionally reset. The maximum range for WD2 is 255 seconds.

The WD1 timer is typically set to a shorter interval than the WD2 timer. User applications can examine the expiration status of the WD1 timer to get advance warning if the main timer, WD2, is about to expire. The system management software has to start WD1 before it can start WD2. If WD1 expires, then WD2 starts only if enabled. The maximum range for WD1 is 6553.5 seconds.

The watchdog subsystem is managed by a PICL plug-in module. This PICL plug-in module provides a set of PICL properties to the system, which enables a Solaris PICL client to specify the attributes of the watchdog system.

To use the PICL API to set the watchdog properties, your application must adhere to the following sequence:

1. Before setting the watchdog timer, use the PMS API to disable the primary HEALTHY# signal monitoring for the CPU board on which the watchdog timer is to be changed.

   To do this, switch to the alarm card CLI and use the command `pmsd infoshow`, specifying the slot number. The output will indicate whether the card is in `MAINTENANCE` mode or `OPERATIONAL` mode.

```
# pmsd infoshow -s slot_number
  config=<MAINTENANCE|OPERATIONAL>
              ALARM_STATE=NONE
```

If the card is in OPERATIONAL mode, switch it into MAINTENANCE mode by issuing the following command:

```
# pmsd operset -s slot_number -o MAINT_CONFIG
```

This disables the primary HEALTHY# signal monitoring of the board in the specified slot.

2. In your application, use the PICL API to disarm, set, and arm the active watchdog timer.

   Refer to the `picld`(1M), `libpicl`(3LIB), and `libpicl`(3PICL) man pages for a complete description of the PICL architecture and programming interface. Develop your application using the PICL programming interface to do the following:

   - Disarm the active watchdog timer.
   - Change the watchdog timer PICL properties to the required values.
   - Re-arm the watchdog timer. The properties of watchdog-controller and watchdog-timer are defined in TABLE 8-11, TABLE 8-12, and TABLE 8-13.

3. Use the PMS API to enable the primary HEALTHY# signal monitoring on the CPU card in the specified slot.

   From the alarm card CLI, switch the card back to operational mode by issuing the following command:

```
# pmsd operset -s slot_number -o OPER_CONFIG
```

HEALTHY# monitoring will be enabled again on the card in the slot that you specified.

Refer to Chapter 7 for information on Processor Management Services (PMS).

PICL interfaces for the watchdog plug-in module (see TABLE 8-11) include the nodes watchdog-controller and watchdog-timer.

TABLE 8-11 Watchdog Plug-in Interfaces for Netra CT 810 and 410 Server Software

PICL Class	Property	Meaning
`watchdog-controller`	`WdOp`	Represents a watchdog subsystem.

TABLE 8-11 Watchdog Plug-in Interfaces for Netra CT 810 and 410 Server Software

PICL Class	Property	Meaning
`watchdog-timer`	`State`	Represents a watchdog timer hardware that belongs to its controller. Each timer depends on the status of its peers to be activated or deactivated.
	`WdTimeout`	Timeout for the watchdog timer
	`WdAction`	Action to be taken after the watchdog expires.

TABLE 8-12 Properties Under `watchdog-controller` Node

Property	Operations	Description
`WdOp`	`arm`	Activates all timers under the controller with values already set for `WdTimeout` and `WdAction`.
	`disarm`	All active timers under the controller will be stopped.

TABLE 8-13 Properties Under `watchdog-timer` Node

Property	Values	Description
`State`	`armed`	Indicates timer is armed or running. Cleared by `disarm`.
	`expired`	Indicates timer has expired. Cleared by `disarm`.
	`disarmed`	Default value set at boot time. Indicates timer is disarmed or stopped.
`WdTimeout`[*]	Varies by system and timer level	Indicates the timer initial countdown value. Should be set prior to arming the timer.
`WdAction`[†]	`none`	Default value. No action is taken.
	`alarm`	Send notifications to system alarm hardware by means of HEALTHY#.
	`reset`	Perform a soft or hard reset the system (implementation specific).
	`reboot`	Reboot the system.

* A platform might not support a specified timeout resolution. For example Netra CT systems only take -1, 0, and 100 6553500 ms in increments of 100 msec. (Level 1), and -1 -255 seconds (Level 2).

† A specific timer node might not support all action types. For example Netra CT watchdog level 1 timer supports only "`none`" and "`alarm`" actions. Watchdog level 2 timer supports only "none" and "reset"

To identify current settings of watchdog-controller, issue the command `prtpicl -v` as shown in CODE EXAMPLE 8-2.

CODE EXAMPLE 8-2 Example of `watchdog-controller`

```
# prtpicl -v
         <snip>
        watchdog-controller1 (watchdog-controller,3600000729)
                :wd-op  disarm
                :_class watchdog-controller
                :name   watchdog-controller1
                   watchdog-level1 (watchdog-timer, 360000073f)
                        :WdAction     alarm
                        :WdTimeout    0x1f4
                        :State        armed
                        :_class       watchdog-timer
                        :name  watchdog-level1
                   watchdog-level2 (watchdog-timer, 3600000742)
                        :WdAction     none
                        :WdTimeout    0xffff
                        :State        disarmed
                        :_class       watchdog-timer
                        :name  watchdog-level2
```

Displaying FRU-ID Data

Sun FRU-ID is the container for presenting the FRU-ID data. If the Sun FRU-ID container is not present, the FRU-ID Access plug-in looks for the IPMI FRU-ID container of cPCI FRUs. It then converts FRU-ID data from IPMI format to Sun FRU-ID format and presents the result in Sun FRU-ID ManR (manufacturer record) format.

The command `prtfru`(1M) displays FRU data of all FRUs in the PICL frutree. When `prtfru` is run on the host CPU, FRU data is displayed for the host CPU and the satellite CPUs.) CODE EXAMPLE 8-3 shows an example of the output of the `prtfru` command.

CODE EXAMPLE 8-3 Sample Output of `prtfru` Command

```
# prtfru
/frutree
/frutree/chassis (fru)
/frutree/chassis/CPU?Label=CPU 1
```

CODE EXAMPLE 8-3 Sample Output of `prtfru` Command *(Continued)*

```
/frutree/chassis/CPU?Label=CPU 1/CPU (container)
   SEGMENT: FD
      /ECO_CurrentR
      /ECO_CurrentR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /ECO_CurrentR/Firmware_Revision: 00000000
      /ECO_CurrentR/Hardware_Revision: 00
      /ECO_CurrentR/HW_Dash_Level: 00
      /CPUFirmwareR
      /CPUFirmwareR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /CPUFirmwareR/CPU_FW_Part_No: 5251979
      /CPUFirmwareR/CPU_FW_Dash_Level: 05
      /Drawer_InfoR
      /Drawer_InfoR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /Drawer_InfoR/Drawer_Id: 000000
      /Drawer_InfoR/Drawer_Type: 0000000000000000
      /Drawer_InfoR/Access_Model: 0000000000000000
      /Drawer_InfoR/Slot_Mode: 0000000000000000
      /Drawer_InfoR/Reserved_Data:
000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000
0000000000000000000000000
      /Customer_DataR
      /Customer_DataR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /Customer_DataR/Cust_Data: Customer Data
   SEGMENT: SD
      /ManR
      /ManR/UNIX_Timestamp32: Thu Jul 11 17:00:00 PDT 2002
      /ManR/Fru_Description: ASSY,CPCI,CP2140,OTHELLO+
      /ManR/Manufacture_Loc: Pleasanton
      /ManR/Sun_Part_No: 5016358
      /ManR/Sun_Serial_No: 999999
      /ManR/Vendor_Name: Hal Computers
      /ManR/Initial_HW_Dash_Level: 08
      /ManR/Initial_HW_Rev_Level: 99
      /ManR/Fru_Shortname: CPU
      /SpecPartNo: XXX-0071-05
/frutree/chassis/CPU?Label=CPU 1/CPU/PMC-1?Label=PMC
/frutree/chassis/AL-8?Label=AL 8
/frutree/chassis/AL-8?Label=AL 8/AL-8 (container)
   SEGMENT: FD
/ECO_CurrentR
      /ECO_CurrentR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /ECO_CurrentR/Firmware_Revision: 00000000
      /ECO_CurrentR/Hardware_Revision: 00
      /ECO_CurrentR/HW_Dash_Level: 00
      /Netra_ACFirmwareR
      /Netra_ACFirmwareR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
```

CODE EXAMPLE 8-3 Sample Output of `prtfru` Command *(Continued)*

```
      /Netra_ACFirmwareR/BCFW_Part_No: 5251971
      /Netra_ACFirmwareR/BCFW_Dash_Level: 05
      /Netra_ACFirmwareR/CMSW_Part_No: 5251972
      /Netra_ACFirmwareR/CMSW_Dash_Level: 05
      /Netra_ACFirmwareR/BMCFW_Part_No: 5251965
      /Netra_ACFirmwareR/BMCFW_Dash_Level: 03
      /Customer_DataR
      /Customer_DataR/UNIX_Timestamp32: Wed Dec 31 16:00:00 PST 1969
      /Customer_DataR/Cust_Data: Customer Data
   SEGMENT: SD
      /SpecPartNo: XXX-0015-05
      /ManR
      /ManR/UNIX_Timestamp32: Thu Jul 11 17:00:00 PDT 2002
      /ManR/Fru_Description: - ASSY,MAC,MAKA8
/ManR/Manufacture_Loc: Pleasanton CA
      /ManR/Sun_Part_No: 5016171
      /ManR/Sun_Serial_No: 999999
      /ManR/Vendor_Name: Hal Computers
      /ManR/Initial_HW_Dash_Level: 05
      /ManR/Initial_HW_Rev_Level: 99
      /ManR/Fru_Shortname: AlarmCard
/frutree/chassis/IO-2?Label=I.O 2
/frutree/chassis/IO-2?Label=I.O 2/IO-2 (container)
   SEGMENT: FD
   SEGMENT: SD
      /ManR
      /ManR/UNIX_Timestamp32: Thu Dec  5 06:19:32 PST 2002
      /ManR/Fru_Description: ASSY,MB,650MHZ,1GB,SPUTNIK+
      /ManR/Manufacture_Loc: MITAC TAIWAN
      /ManR/Sun_Part_No: 3753129
      /ManR/Sun_Serial_No: 000355
      /ManR/Vendor_Name: Mitac International
      /ManR/Initial_HW_Dash_Level: 02
      /ManR/Initial_HW_Rev_Level: 01
      /ManR/Fru_Shortname: CPU
      /SpecPartNo: 885-0111-02
/frutree/chassis/IO-3?Label=I.O 3
/frutree/chassis/IO-3?Label=I.O 3/IO-3 (container)
   SEGMENT: FD
   SEGMENT: SD
      /ManR
      /ManR/UNIX_Timestamp32: Thu Dec  5 05:25:03 PST 2002
      /ManR/Fru_Description: ASSY,MB,650MHZ,1GB,SPUTNIK+
      /ManR/Manufacture_Loc: MITAC TAIWAN
      /ManR/Sun_Part_No: 3753129
      /ManR/Sun_Serial_No: 000245
      /ManR/Vendor_Name: Mitac International
```

CODE EXAMPLE 8-3 Sample Output of `prtfru` Command *(Continued)*

```
      /ManR/Initial_HW_Dash_Level: 02
      /ManR/Initial_HW_Rev_Level: 01
      /ManR/Fru_Shortname: CPU
      /SpecPartNo: 885-0111-02
/frutree/chassis/IO-4?Label=I.O 4
/frutree/chassis/IO-4?Label=I.O 4/IO-4 (fru)
/frutree/chassis/IO-5?Label=I.O 5
/frutree/chassis/IO-6?Label=I.O 6
/frutree/chassis/IO-7?Label=I.O 7
/frutree/chassis/RTM?Label=RTM
/frutree/chassis/c0::dsk.c0t5d0?Label=RMM
/frutree/chassis/c1::dsk.c1t1d0?Label=HDD 1
/frutree/chassis/c1::dsk.c1t1d0?Label=HDD 1/c1::dsk.c1t1d0 (fru)
/frutree/chassis/c0::dsk.c0t0d0?Label=HDD 0
/frutree/chassis/c0::dsk.c0t0d0?Label=HDD 0/c0::dsk.c0t0d0 (fru)
```

MCNet Support

Communication between CPUs is enabled by MCNet (`mcn`(7D)), which presents an Ethernet-like interface over the cPCI bus in accordance with PICMG 2.14. The interface is configured automatically during system boot, and supports all existing network tools, such as `ifconfig`(1M), `netstat`(1M) and so forth. The CPUs must be MCNet-capable in order to communicate with each another.

Glossary

A

AC	Alarm card. The alarm card is used in the Netra 810 and Netra CT 410 servers to provide system control functions. The alarm card resides in slot 8 in the Netra CT 810 server and in slot 1 in the Netra 410 server.
ACL	Access control list; a file that details which SNMP management applications can access information maintained by the MOH. The file also lists which hosts can receive SNMP traps or events.
Alarm Severity Profile	A managed entity that contains the severity assignments for the reported alarms.
ASN1	Abstract notation one. The notation used in a text file for a MIB.The variables containing the information that SNMP can access are described in this file.
Attribute Value Change Record	A managed entity used to represent logged information resulting from attribute value change notifications. Instances of this managed entity are created automatically by the network entity (NE), and deleted by the NE or by request of the managing system.

C

CGTP	Carrier Grade Transport Protocol. CGTP network interfaces send and receive packets on redundant networks. These software devices use CGTP protocol. See the `ifcgtp`(7) man page, which details the general properties of the network interfaces.
CLI	Command-line interface. The primary user interface to the alarm card.
cPCI	Compact PCI.

E

EFDMBean	Event Forwarding Discriminator. A managed entity used as a notification forwarder discriminator. At startup it registers itself as a listener to all the broadcaster MBeans registered with the MBeanServer, then listens for MBeanServer creation notifications to register with newly created MBeans.
Equipment	A managed entity used to represent the various externally manageable physical components of the network entity (NE) that are not modeled using the Plug-in Unit or Equipment Holder managed entities.
Equipment Holder	A managed entity representing physical resources of the NE that are capable of holding other physical resources. An instance of this managed entity exists for each rack, shelf, drawer, and slot of the NE.

F

Full Log	A managed entity used to group multiple instances of the Managed Entity Creation Log Record, Managed Entity Deletion Log Record, State Change Log Record, Attribute Value Change Log Record, and/or Alarm Record managed entities to form a log. This managed entity contains information that, among other things, allows the management system to control the behavior of the log.

G

GPIO	General purpose I/O.

H

Host	Host CPU board. In the Netra CT 810 server, the host CPU board resides in slot 1. In the Netra CT 410, the host CPU board resides in slot 3.

I

IM	Information model
IPMI	Intelligent Platform Management Interface, used as a communication channel over the cPCI backplane in the Netra CT server.

L

Latest Occurrence Log	A managed entity used to group multiple log records to form a latest occurrence log. If no other log record contained in the Latest Occurrence Log instance has values of the attributes identified by the Key Attribute List attribute equal to the attribute values of the log record to be added, the log record is created and contained in the Latest Occurrence Log.

M

MCNet	A communication channel running over the CompactPCI backplane. It can be used to communicate between the alarm card, the host CPU board, and any satellite boards.

MIB	Managed information base used to describe the exchange of information across the network element (NE) interface. A MIB is loadable, but can reference other MIBs.
module	Software modules are part of a program that are not combined with other parts until the program is linked. Modules do not have to be changed when a new type of object is added.
MOH	Managed Object Hierarchy. An application that monitors the field replaceable units in the system. MOH runs on the alarm card, the host CPU, and any satellite CPUs.

N

NE	Network Element Managed Entity. A component of the MIB. An instance of this managed entity is automatically created upon initialization
NFS	Network File System.
NIS	Network Information System.

P

Physical Path Termination Point	See Termination Point MBean.
PICL	Platform Information Configuration Library. A Solaris OS library that provides a method used to publish platform-specific information for clients to access in a way that is not specific to the platform.
Plug-in Unit	A managed entity used to represent equipment that is inserted (plugged into) and removed from slots of the NE.
PMS	Processor Management Service. Manages processor elements used by client applications to implement high availability.
POSIX	IEEE version of UNIX first published in 1968. The latest version now merges with The Open Group's Specification which comprise the core of the Single UNIX Specification.

R

RCM	Reconfiguration Coordination Manager. Part of the Solaris OS's dynamic reconfiguration (DR) framework that enables automated DR removal operations on platforms with appropriate software and hardware configuration.
RDHCP	Reliable Dynamic Host Configuration Protocol.
RMI	Remote Method Invocation. Java RMI is a mechanism that allows one to invoke a method on an object that exists in another address space.
RNFS	Reliable Network File System.

S

SAT	Satellite. An auxiliary CPU board that occupies a designated cPCI slot on the Netra CT system, which might, under certain conditions, operate independently.
SNMP	Simple Network Management Protocol. A protocol that allows devices to be controlled remotely by a network management station.
SMI	Structure of Management Information. A definition that describes the syntax and basic data types available in a given MIB.
Software MBean	A managed entity representing logical information stored in equipment, including programs and data tables. Instances of this managed entity are created by the NE to report to the management system, the currently installed software in the related entity (that is, NE, equipment or Plug-In Unit).
State Change Record	A managed entity used to represent logged information resulting from state change notifications. Instances of this managed entity are created automatically by the NE, and deleted by the NE or by request of the managing system.

T

TFTP	Trivial File Transfer Protocol.

Termination Point MBean	A managed entity used to represent the points in the NE where physical paths terminate (such as ports), and physical path level functions (for example, path overhead functions) are performed.
Topology Change Notification	An abstract class representing generic notifications for a change in the topology of a network entity.

Index