Netra™ CT 820 Server Software Developer's Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 817-2648-11
May 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 820 Server Software Developer's Guide* contains information for developers writing application software for the Netra™ CT 820 server. 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 820 server 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 820 server software views the hardware components.

Chapter 3 offers a tutorial in writing applications that interface with the Netra CT 820 server software.

Chapter 4 introduces the application programming interfaces for the Netra CT 820 server including the Netra CT Element Management (`netract`) agent software.

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

Chapter 6 provides an overview of the Netra CT Processor Management Service (PMS) software.

Chapter 7 defines the Solaris™ Operating System's platform information and control library (PICL) software and how you can use it to set the watchdog timer.

The Glossary contains a list of for special terms and their definitions.

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 820 server documentation is listed in the following table.

Title	Part Number
Netra CT 820 Server Product Overview	817-2643
Netra CT 820 Server Installation Guide	817-2641
Netra CT 820 Server Service Manual	817-2642
Netra CT 820 Server System Administration Guide	817-2647
Netra CT 820 Server Safety and Compliance Manual	817-2645
Netra CT 820 Server Software Developer's Guide	817-2648
Netra CT 820 Server Product Note	817-2646

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

Accessing Sun Documentation

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

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

`docfeedback@sun.com`

Please include the part number (817-2648-11) of your document in the subject line of your email.

CHAPTER 1

Programming Environment

This chapter provides an overview of the software environment that forms the basis for developing applications for the Netra CT 820 server:

Netra CT 820 Server

The Netra CT 820 server system includes two Distributed Management Cards (DMCs) providing the nexus of system management, up to 18 node cards, and two switch fabric cards which are linked to each node card through a packet-switching midplane. Different software combinations run on each of these elements as is shown in FIGURE 1-1.

Hardware Descriptions

This section describes card components of the Netra CT 820 server.

Distributed Management Card (DMC)

The two DMCs in the Netra CT 820 server control the system functions, including system power and cooling. These 3U cards are plugged into slot 1 (in slot 1A and slot 1B), and are essentially special purpose single card computers, capable of

querying the status of system elements, as well as configuring and controlling the power-up sequence of each device. ChorusOS 5.0 is the operating system running on each distributed management 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. Monitoring and control of the system is accomplished through Managed Object Hierarchy (MOH) and Processor Management Service (PMS) software.

Switching Fabric Card

An IPMI-supported switching fabric card links to each node card through a packet switching midplane at 100 Mbps and 1000 Mbps. This enables each node card to communicate with every other node card thus creating a packet- switching fabric. A switching fabric card can only occupy either slot 2 or 21, which are dedicated to PICMG 2.16-compliant switching fabric cards. The two switching fabric cards in the Netra CT 820 server are controlled by a node CPU and the Solaris Operating System (OS) running on that card.

Node Cards

Each of 18 possible node cards is linked to both switching fabric cards through the Netra CT 820 midplane. The node cards used in the Netra CT 820 are the Netra CP2300 cPSB cards, which are plugged into node slots 3-20. The Solaris OS runs on the CP2300 cPSB cards. MOH and PMS are provided for local monitoring, although most of the drawer-level monitoring and control functions occur through the DMC.

Besides the CP2300 cPSB card, there are third-party PMC cards that are qualified as node cards for the Netra CT 820 server.

Hot-swapping

Cards and other field-replaceable units (FRUs) can be swapped while the system is running, depending on whether or not they conform to Hot Swap Specification PICMG 2.16. This ability to hot-swap is a feature that is controllable by software if the card itself is hot-swap compliant. For further information on hot-swap issues, see the *Netra CT 820 Server Installation Guide* (817-2641).

Software Descriptions

This section provides an overview and brief descriptions of the Netra CT 820 server software shown in FIGURE 1-1. The abbreviations shown in FIGURE 1-1 are identified in TABLE 1-1.

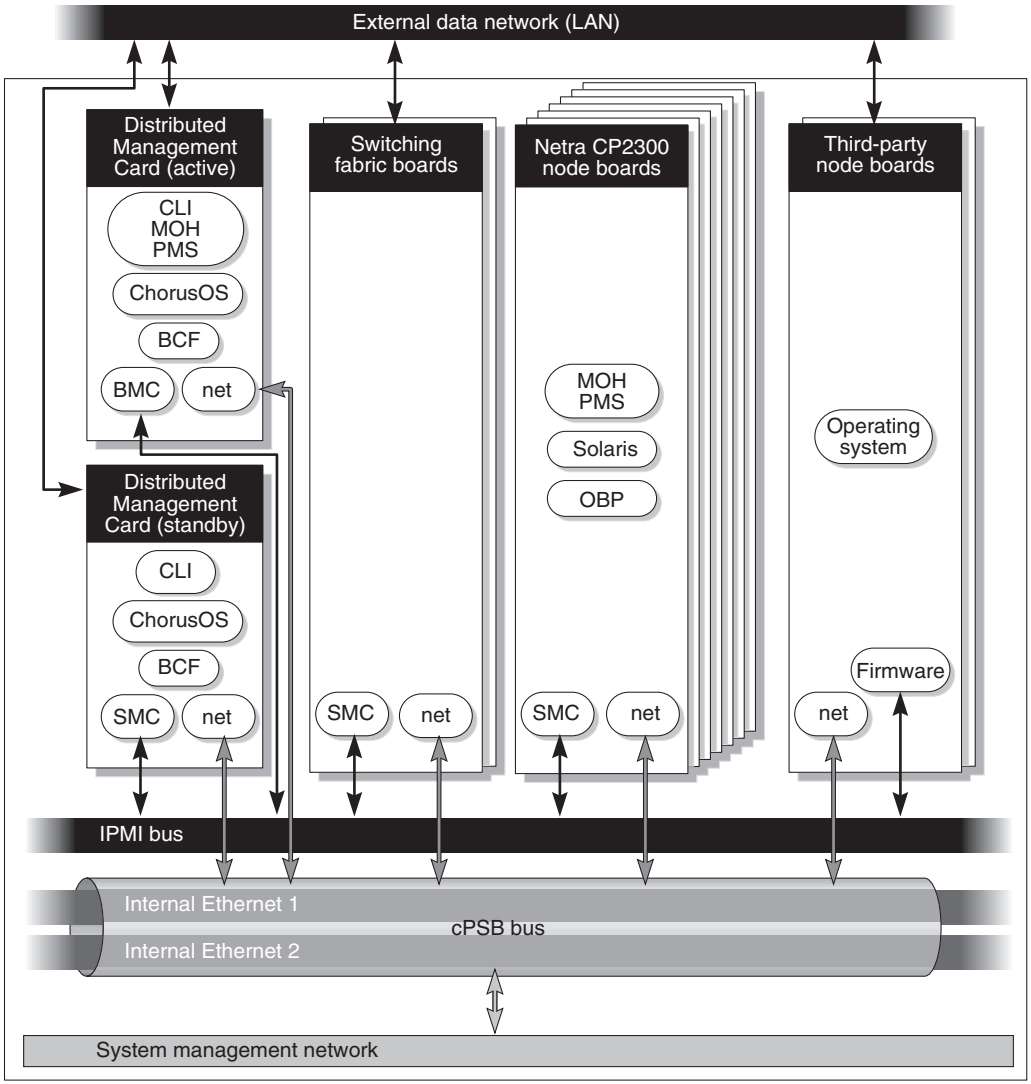


FIGURE 1-1 Netra CT 820 Server Software

TABLE 1-1 Netra CT 820 server Software Overview

Abbreviation	Name	Description
Solaris	Solaris Operating System	Installed by the user. Runs on the node cards.
ChorusOS	ChorusOS operating environment	Factory-installed on the distributed management card. Manages all elements of the Netra CT 820 server that are connected to the midplane.
CLI	Command-line interface	The primary user interface to the distributed management card.
cPSB	CompactPCI Packet Switching Backplane	A packet-based switching architecture on top of CompactPCI.
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 node cards.
BCF	Boot control firmware	Firmware on the distributed management card to control booting.
BMC	BMC firmware	Baseboard management controller of the IPMI Controller on the distributed management card, which provides a command nexus between node CPU and remote management card (RMC) client during hot swap unconfiguration operations.
SMC	SMC firmware	System management controller firmware is related to IPMI Controller on node cards. SMC APIs provide client access to local resources such as temperature sensors, watchdog subsystems, and local I^2C bus devices; and access to IPMI bus devices.
IPMI	IPMI	Intelligent platform management interface is a communication channel over the compactPCI backplane.

Solaris Operating System

Solaris Operating System on the node cards provides APIs such as platform information and control library (PICL), Reconfiguration Coordination Manager (RCM), and the configuration administration utility (`cfgadm` (1M)), as explained in Chapter 7. The kernel layer interacts with device drivers to control hardware components of the system. These device drivers bind to the kernel using the device driver interfaces (DDI) and driver kernel interfaces (DKI).

ChorusOS

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

Managed Object Hierarchy

The Managed Object Hierarchy (MOH) is a distributed management application that runs on the distributed management card and node CPU cards. MOH on the distributed management card provides drawer-level monitoring of the system. MOH on the node CPUs provides local views of the card on which it runs, and collaborates to provide the status of its components to the MOH on the distributed management card. The various MOHs communicate with one another over the packet-switching midplane. The MOH API is discussed in Chapter 4.

Processor Management Services

Processor management services (PMS) software is an extension to the Netra CT 820 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 card elements within a single Netra CT server or within a cluster of multiple Netra CT servers.

PMS ensures high availability by monitoring a processor element's fault condition, such as OS hangs, deadlock, and panic. The distributed management card provides a server-level view showing the state of each node card as a plug-in unit. PMS services are enabled separately on the distributed management card and on the node CPU. PMS services are discussed further in Chapter 6.

Platform Information and Control Library

The Platform Information and Control Library (PICL) is a Solaris library that facilitates a method for publishing platform-specific information that client applications can access in a way that is not specific to the platform. PICL is discussed further in Chapter 7.

Management Framework

The Java™ Dynamic Management Kit (DMK) 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 the Netra CT 820 system.

SNMP/MIB Support

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

- System group from MIB II
- Interface group from the IF-MIB
- Physical entity group from the ENTITY-MIB

SNMP Interface

The `netract` agent operates on the distributed management card and the system node cards in a distributed manner. They all provide the SNMP interface version 2, and Netra CT-specific instrumentation monitoring.

RMI Interface

The Netra CT Management Agent uses Java DMK 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 distributed management card and node CPUs. To develop applications that use PMS, you need Solaris OS, C compiler, PMS API, and libraries as described in Chapter 6.

Developing Applications to Interface With MOH or SNMP

To develop applications to interface with MOH or SNMP, you need the Solaris OS, Java virtual machine (JVM)[1], Java DMK and the Netra CT agent library. For more information about Java DMK, refer to *Java Dynamic Management Kit 4.2 Tutorial* (806-6633).

Developing Applications to Run on Node Cards

To develop applications to run on node cards you require Solaris OS to access services such as dynamic reconfiguration (DR) framework, and Platform Information and Control Library (PICL) API.

1. The terms "Java virtual machine" and "JVM" mean a virtual machine for the Java platform.

CHAPTER 2

Netra CT 820 System Equipment Model

This chapter provides illustrations of the Netra CT 820 server equipment models and contains the following sections:

Modeling a Netra CT 820 System

Equipment models show how the Netra CT Element Management Agent software views the Netra CT 820 server hardware. Each equipment model presents a Netra CT 820 server in a containment hierarchy of hardware components, with the midplane at the root of the hierarchy. For example, a compactPCI packet switching backplane (cPSB) slot can contain a distributed management card, which in turn contains 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.

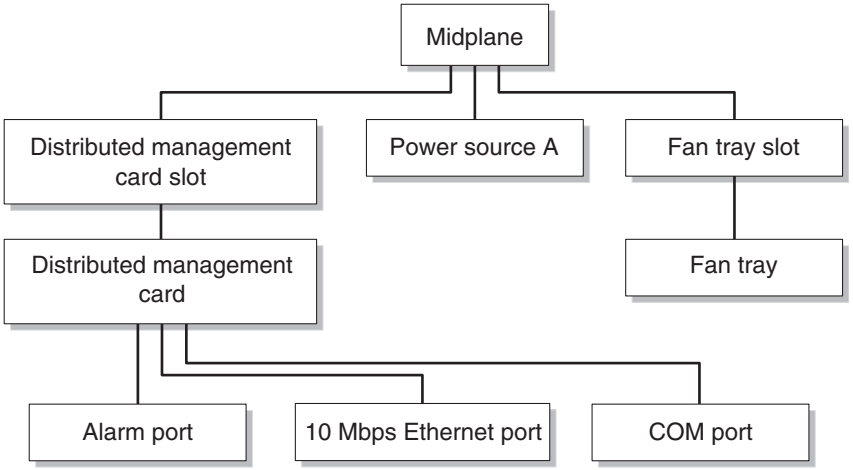


FIGURE 2-1 Segment of Hardware Resource Hierarchy

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 might be represented by a single managed object or by several managed objects. An agent typically provides views of many managed objects.

FIGURE 2-2 shows the class names of the Netra CT software managed objects that refer to hardware, and TABLE 2-1 defines these objects.

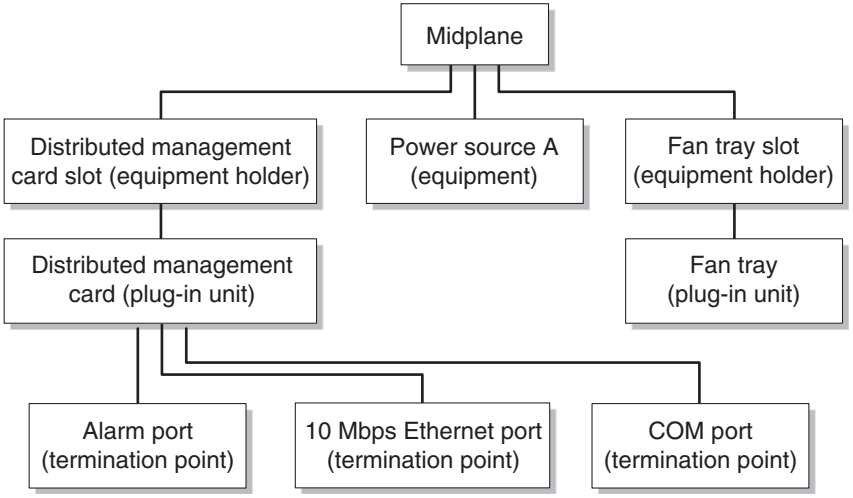


FIGURE 2-2 Hardware Resource Hierarchy Showing Managed Object Classes

TABLE 2-1 Managed Object Class Definitions

Managed Object Class	Definition
Network element	May be standalone devices or multicomponent, geographically distributed systems.
Equipment holder	Represents physical resources of the network element that are capable of holding other physical resources. For example, cPSB slots, fan tray slots, and switching fabric board slots are equipment holder resources.
Plug-in unit	Represents equipment that can be physically inserted or removed from slots of the system (for example, node cards and power supply units).
Equipment	Represents those externally manageable physical components which are not FRUs (for example, a CPU thermistor or fan tray 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 Hierarchy

Both the SNMP interface and the Java Management Extensions (the JMX™ specification) are compatible with Netra CT element management API, and provide ways to traverse the equipment containment hierarchy. You can view the managed objects of a Netra CT 820 server through the system's distributed management card. You can also view the managed objects from the agent on any node card. 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 distributed management card (defined as the *system view from the distributed management card*), the distributed management card's termination points (alarm port, Ethernet port, serial, and COM ports) are displayed in the model, but a node card's termination points are not displayed.

You can also view the equipment model with the node card as the network element at the top of the hierarchy. In these models (defined as the *node card local view*), only the objects directly controlled by the node card are displayed. Other objects, like the midplane, distributed management card, and the power distribution unit, are not displayed in these equipment models.

"Netra CT 820 System Equipment Models" on page 13 presents the equipment model for the Netra CT rear-access system.

Netra CT 820 System Equipment Models

FIGURE 2-3 shows the local view through the node card and FIGURE 2-4 shows the system-level view through the distributed management card. In the latter figure, RTM refers to the rear transition module.

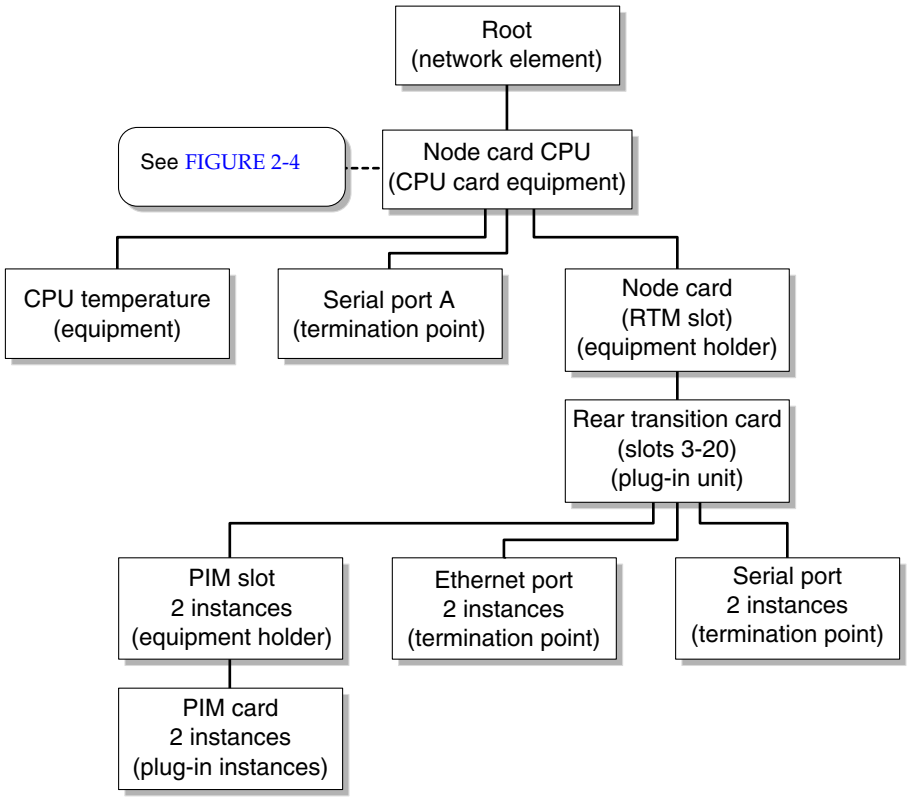
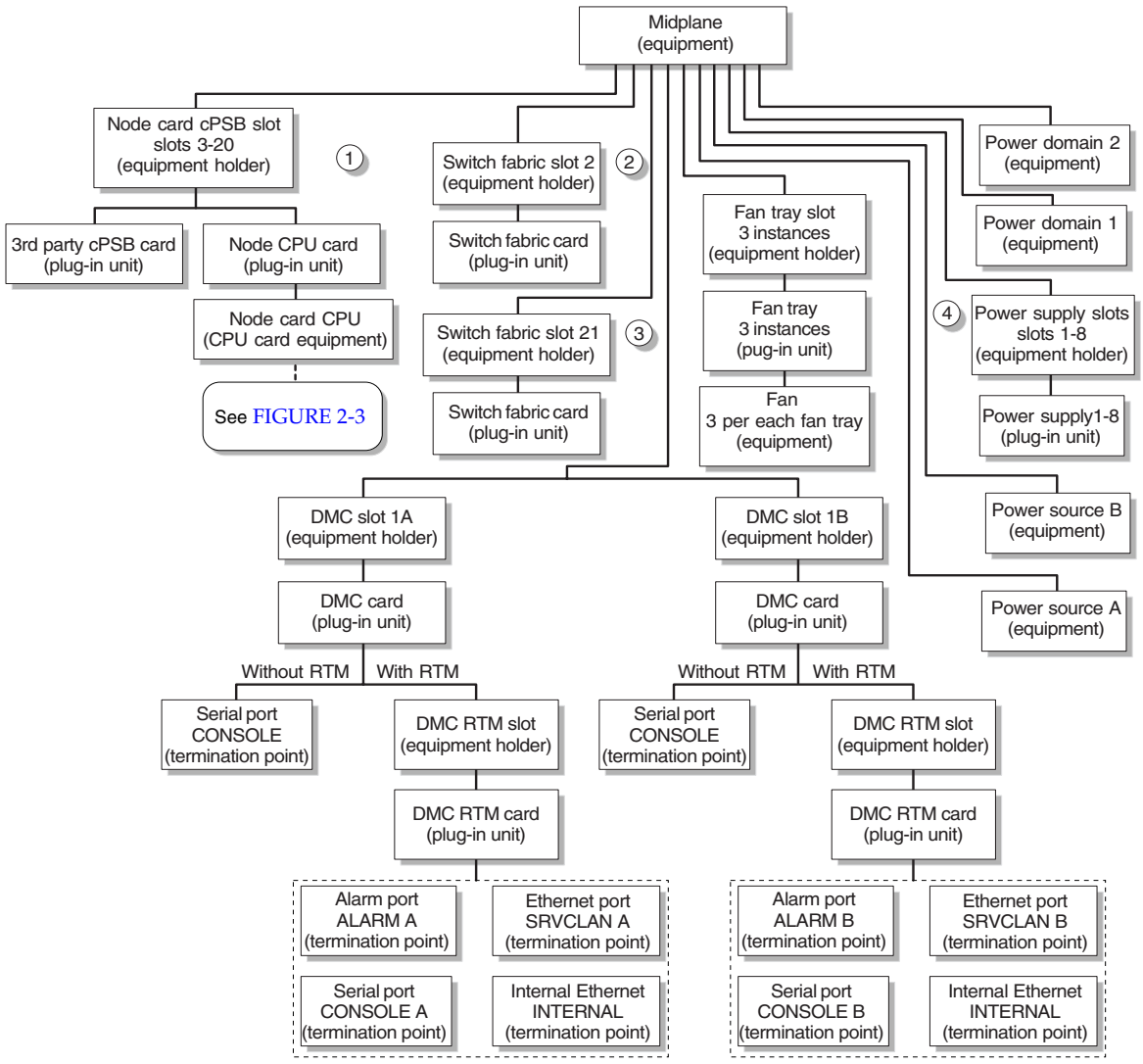


FIGURE 2-3 Netra CT 820 System Model Node Card Local View



Number 1-4 indicates the association of related objects to power domain and/or power source	Association relationship
1) cPSB slots 3-11	Power domain 1
cPSB slots 12-20	Power domain 2
2) Switch fabric slot 2	Power domain 1
3) Switch fabric slot 21	Power domain 2
4) Power supply slots 1, 3, 5, 7	Power source A
Power supply slots 2, 4, 6, 8	Power source B
Power supply slots 1, 2, 3, 4	Power domain 1
Power supply slots 5, 6, 7, 8	Power domain 2

FIGURE 2-4 Rear-Access Netra CT 820 System View From Distributed Management Card

CHAPTER 3

Getting Started With the Netra CT Element Management Agent API

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

Before You Begin

You should become acquainted with the topology of the Netra CT 820 server (see Chapter 2), and have some knowledge of Java programming, JMX specifications, and the Java Dynamic Management Kit (JDMK) framework. For more information about JDMK refer to the *Java Dynamic Management Kit 4.2 Tutorial* (806-6633) available at `http://docs.sun.com/db/doc/806-6633`.

Verify that you have the Solaris OS and the packages listed in TABLE 3-1 installed on your development system. You will use these installed packages to work with this tutorial.

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`	Distributed management card firmware package that includes the Netra CT Management Agent

Note – These packages are installed as part of the Netra CT software. Refer to the *Netra CT 820 Server Release Notes* (817-2646) for information on required Netra CT 820 patches.

About Netra CT Element Management Agent API

The Netra CT server software package includes various modules and extensions (see "Solaris Operating System" on page 5), 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 distributed management cards (DMC), and node CPU cards. 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 management applications, one is the SNMPv2C interface, the other is a JMX-compatible Java API called the Netra CT Management Agent API (or `netra` agent API). This chapter provides an introduction on how to write a management application using the JMX-compatible `netra` agent API.

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:

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

The following section summarizes the procedure for putting together a management application for the Netra CT server.

Focus 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 boards are in which slots, and so on. The section, "To Determine the System Configuration Hierarchy" on page 18, shows code that starts this action.

Next, the application should determine how to monitor event notifications. The section, "Listening for Notifications" on page 21, continues with developing a way of monitoring notifications such as power on and power off to a particular slot or device.

After event monitoring, the application needs to manage the system alarms. "Managing Alarms" on page 23 describes alarm management: how to handle the receiving and transmitting of system alarms such as CPU over-temperature alarm.

The following steps give you an overview of the procedure involved in putting together an application.

▼ To Prepare Your Application

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* (806-6633) for background information on `Client.java`.

2. **To compile** `Client.java`**, issue the following command:**

```
$ /usr/j2se/bin/javac -classpath \
/opt/SUNWjdmk/jdmk4.2/1.2/lib/jdmkrt.jar:\
/opt/SUNWnetract/mgmt3.0/lib/agent.jar Client.java
```

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

3. **Before running** `Client.java`**, start the agent by issuing the following command:**

```
$ /opt/SUNWnetract/mgmt3.0/bin/ctmgx start
```

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

```
$ /usr/j2se/bin/java -classpath \
.:/opt/SUNWjdmk/jdmk4.2/1.2/lib/jdmkrt.jar:\
/opt/SUNWnetract/mgmt3.0/lib/agent.jar Client
```

▼ To Determine the System Configuration Hierarchy

In this section you develop a client to print out the object names of the management beans (MBeans) representing the system. The sample code is separated into three parts:

Part 1: Communicating With the Netra CT Agent

Part 2: Finding the Root Object Name

Part 3: Traversing the Containment Hierarchy From a Node

A complete description of MBeans, together with examples, can be found in the Java DMK documentation.

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

Refer to the *Netra CT 820 Server System Administration Guide (817-2647)* if you need help to install the appropriate software.

2. **Go to the on-line API documentation and lookup the documents that will identify the elements needed to communicate with the Netra CT agent.**

   See the Netra CT Management Agent API documentation at `/opt/SUNWnetract/mgmt3.0/docs/api/index.html` for the following:

   - `com.sun.ctmgx.MohNames`
   - `com.sun.ctmgx.ContainmentTreeMBean`

   For the JDMK documentation, go to `/opt/SUNWjdmk/jdmk4.2/1.2/docs`. For an introduction to JDMK, see the *Java Dynamic Management Kit 4.2 Tutorial* at `http://docs.sun.com/db/doc/806-6633`.

   See the JDMK API documentation for:

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

Communicating With the Netra CT Agent

This simple demonstration enables you to connect a client with an instance of `netract` agent, beginning in CODE EXAMPLE 3-1. This example represents part one of a three-part example. A detailed explanation follows.

CODE EXAMPLE 3-1 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;
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 {
```

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

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

CODE EXAMPLE 3-1 instantiates the **RmiConnectorClient** and **RmiConnectorAddress**.

The demonstration continues in CODE EXAMPLE 3-2.

Finding the Root Object Name

CODE EXAMPLE 3-2 continues from the previous example. The code snippet connects to the client and displays the ContainmentTree by getting the ObjectName of the root MBean in the containment hierarchy.

Each **MohNames** instance reveals **ObjectNames** instances that are accessible through 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 3-2 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];

      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 3-3 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 3-3 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 with "`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 continues from the previous three-part example.

In the JDMK API documentation, look at the following:

- `javax.management.Notification`
- `javax.management.NotificationListener`

- javax.management.NotificationFilterSupport
- javax.management.NotificationFilter

In the Netra CT Management Agent API documentation at /opt/SUNWnetract/mgmt3.0/docs/api/index.html, look at the documentation for the following:

- com.sun.ctmgx.moh.MohNames
- 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 3-4 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 3-4 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 DMC does not support the PUSH_MODE, so the preceding code will work for any of the netract agent instances (those on the node, and DMC) in a Netra CT drawer.

Managing Alarms

Before you begin this segment of code examples, refer back to the Netra CT Management Agent API document at `/opt/SUNWnetract/mgmt3.0/docs/api/index.html` and look at the documentation for the following:

- `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 3-5 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)
            }
 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 3-5 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 starts 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 3-6 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
                // 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];
```

CODE EXAMPLE 3-6 Using the Default Alarm Severity Profile *(Continued)*

```
                    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 3-6, 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 3-7.

CODE EXAMPLE 3-7 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
   //
            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;
```

CODE EXAMPLE 3-7 Creating an Alarm Severity Profile *(Continued)*

```
            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 3-7 assigns alarm notifications for high memory usage, fan failure, and fuse failure, although the current `netract` agent does not support alarm notifications for fan failure or fuse failure. The code snippet is included here for demonstration purposes.

Assigning a New Alarm Severity Profile

CODE EXAMPLE 3-8 shows how to assign a new alarm severity profile to an MBean which can generate AlarmNotifications.

CODE EXAMPLE 3-8 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 DMC Alarm Outputs

The system configuration hierarchy indicates the physical alarm port corresponds to a termination point shown in FIGURE 2-4, "Rear-Access Netra CT 820 System View From Distributed Management Card" on page 14. It 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 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" in the Netra CT Management Agent API document at `/opt/SUNWnetract/mgmt3.0/docs/api/index.html`.

You can configure a DMC agent to drive output alarms from the DMC on the Netra CT 820 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 DMC 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 23, 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 3-9 extracted from "Using the Default Alarm Severity Profile" on page 24, the severity level of `HIGH.TEMPERATURE` AlarmType in the default ASP has been set to `CRITICAL` corresponding with alarm0.

CODE EXAMPLE 3-9 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 3-10 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 3-10 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
```

CODE EXAMPLE 3-10 Extract of Assigning a New Alarm Severity Profile *(Continued)*

```
         // MBean instance
         //
            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`.

CHAPTER 4

Netra CT Element Management Agent Application Programming Interfaces

This chapter describes the application programming interfaces (APIs) 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.

Java DMK supports the Java Management Extensions (JMX) architecture, which is a standard set of APIs for network/client management. Java DMK 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 Java DMK, tutorials, code samples, and APIs on the Java Developers web site:

`http://developer.java.sun.com`.

Summary of Java Dynamic Management Kit

Java Dynamic Management Kit (DMK's) API and development tools can help you develop distributed management applications. The Java DMK enables resources of one host to be monitored from another host.

A *resource* can be any entity, physical or virtual, that you want 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.

For a management agent to monitor a resource, the resource must be developed as a managed bean (MBean), which is a 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 Java DMK 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 want 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 enables agent services to interact with MBeans.

The Java DMK provides protocol connector interfaces that enable remote applications to access agent applications and their resources. Remote Method Invocation (RMI) and HTTP are two such Java DMK 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.

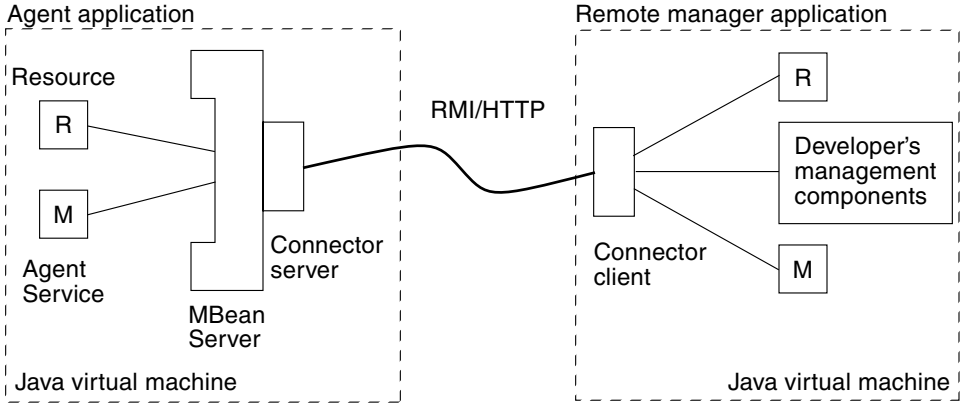


FIGURE 4-1 Key Components of the Java Dynamic Management Kit

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 Java DMK components and protocols, refer to the Java Dynamic Management Kit documentation set found on the Solaris documentation web site, `http://docs.sun.com`. For additional information of Java DMK and the RMI/HTTP protocol, refer to the documentation, tutorials, code samples, and APIs found on the Java Developers web site: `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/mgmt3.0/docs/api/com/sun/ctmgx/moh
```

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

```
file:///opt/SUNWnetract/mgmt3.0/docs/api/index.html
```

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

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

Netra CT Management Agent Interfaces and Classes

TABLE 4-1 lists the Netra CT 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 enables exposing (allowing access to) or hiding (denying access to) an object's access methods, which provides for greater modularity.

Note – The detailed descriptions for the Netra CT Management Agent Interfaces and Classes listed in the tables are in the on-line API documentations at `/opt/SUNWnetract/mgmt3.0/docs/api/`. You can view an index of the API by opening the following link in your web browser:
`file:///opt/SUNWnetract/mgmt3.0/docs/api/index.html`

TABLE 4-1 Netra CT Management Agent Interfaces

Interfaces	Description
`AlarmCardPluginMBean`	Describes the management interface of the AlarmCardPluginMBean.
`AlarmSeverityProfileMBean`	Describes the management interface of the AlarmSeverityProfileMBean.
`ContainmentTreeMBean`	Describes the management interface of the ContainmentTreeMBean.
`CpiSlotMBean`	Describes the management interface of the CPCI Slot objects.
`CpuCardEquipmentMBean`	Describes the management interface of the CPUCardEquipmentMBean.
`CpuPluginMBean`	Describes the management interface of the node card objects as perceived from the DMC MOH.
`EFDMBean`	Describes the management interface of the EFDMBean.
`EquipmentHolderMBean`	Describes the management interface of the EquipmentHolderMBean.
`EquipmentMBean`	Describes the interface of the EquipmentMBean.

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

Interfaces	Description
`FullLogMBean`	This class describes the management interface of FullLog MBean.
`LOLMBean`	This class describes the management interface for Latest Occurrence Log MBean.
`NEMBean`	Describes the management interface of the NEMBean.
`NetworkInterfaceMBean`	Describes the management interface for NetworkInterfaceMBean.
`NodeCardSlotMBean`	Describes the management interface for the node card slot objects.
`NumericSensorMBean`	Describes the interface for NumericSensorMBean.
`PlugInUnitMBean`	Describes the management interface of the Plug-inUnitMBean.
`PowerSupplySlotMBean`	Describes the management interface of the power supply slot objects.
`SensorMBean`	Describes the interface of the SensorMBean.
`SlotMBean`	Describes the management interface of the SlotMBean.
`TerminationPointMBean`	Describes the management interface of the TerminationPoint MBean.

TABLE 4-2 Netra CT Management Agent Classes

Class Summary	Description
`AdministrativeState`	Defines the administrative state of the device.
`AlarmNotification`	The Alarm Notification class represents an alarm notification emitted by an MBean.
`AlarmNotificationFilter`	This filter enables you to filter AlarmNotification notifications by selecting the types and severities of interest.
`AlarmSeverity`	Defines the alarm severity objects for use with Alarm Notification.
`AlarmType`	This class is an enumeration of predefined Alarm types, user need to use one of the predefined types to construct an AlarmNotification object.

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

Class Summary	Description
`AttributeChangeNotification`	Provides definitions of the attribute change notifications sent by MBeans.
`AttributeChangeNotificationFilter`	The filtering is performed on the name of the observed attribute.
`AvailabilityStatus`	Defines the availability status of the plug-in unit object.
`EquipmentHolderType`	Describes the management interface of the EquipmentHolderType.
`LogFullAction`	This class defines the action to perform when the log is full.
`MohNames`	Defines the public constants or static variables for MOH user to communicate to the MBean server.
`ObjectCreationNotification`	Defines the creation notifications sent by MBeans.
`ObjectDeletionNotification`	Defines the deletion notifications sent by MBeans.
`OperationalState`	Defines the operation states of a device (equipment or plug-in).
`SlotStatus`	Defines the status of the slot object.
`StateChangeNotification`	Defines the state change notifications sent by MBeans.
`StateChangeNotificationFilter`	Describes the filtering performed on the name of the observed attribute.

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 string*s. 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`

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 *entries,* 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.

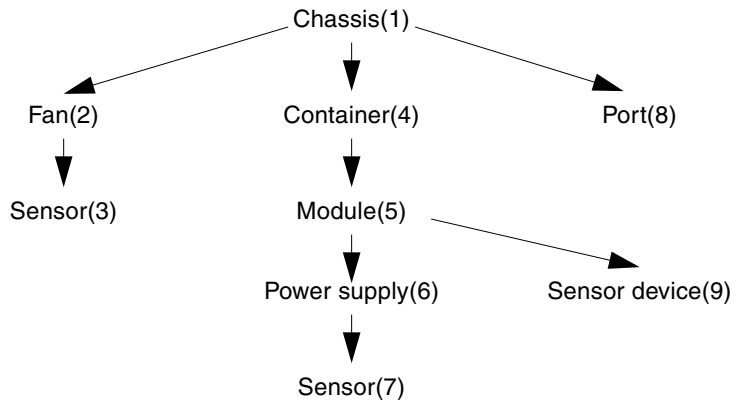


FIGURE 5-1 Hardware Resource Hierarchy

TABLE 5-1 Physical Entity Table Example

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 can 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 42
- "Physical Path Termination Point Interfaces" on page 43
- "Equipment" on page 44
- "Plug-In Unit" on page 45
- "Hardware Unit to Running Software Relationship" on page 46
- "Hardware Unit to Installed Software Relationship" on page 46

Detailed descriptions of the objects are in the MIB file, which is available as part of the software package. As an example of the elements within the file, details follow.

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:

TABLE 5-2 `SUN-SNMP-NETRA-CT-MIB` Netra CT NE 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.
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.
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 Interfaces

The SUN-SNMP-NETRA-CT-MIB module representation of physical path termination point interfaces is composed of the elements shown in TABLE 5-3.

TABLE 5-3 Physical Path Termination Point Interfaces of SUN-SNMP-NETRA-CT-MIB

Field	Description
Physical Path Termination Point Table	The Netra CT Physical Path Termination Point Table augments the entPhysicalTable.
Physical Path Termination Point Table Entry	An entry in the Netra CT Physical Path Termination table. Each entry of this table represents a Physical Path Termination Point within the Netra CT NE.
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.

Equipment

The SUN-SNMP-NETRA-CT-MIB module representation of equipment is composed of the elements shown in TABLE 5-4.

TABLE 5-4 SUN-SNMP-NETRA-CT-MIB Equipment

Field	Description
Equipment Table	The Netra CT Equipment table augments the entPhysicalTable.
Equipment Entry	An entry in the Netra CT Equipment table. 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.
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.
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.
Equipment Holder Table	The Netra CT Equipment Holder table augments the entPhysicalTable.
Equipment Holder Entry	An entry in the Netra CT Equipment Holder table. Each entry in this table represents a component within the Netra CT NE that accepts a replaceable plug-in unit.
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.

TABLE 5-4 SUN-SNMP-NETRA-CT-MIB Equipment *(Continued)*

Field	Description
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

The SUN-SNMP-NETRA-CT-MIB Plug-In Unit table augments the entPhysicalTable and is composed of the elements shown in TABLE 5-5.

TABLE 5-5 SUN-SNMP-NETRA-CT-MIB Plug-In Unit

Field	Description
Plug-In Unit Entry	An entry in the Netra CT Plug-In Unit table. 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.
Plug-In Unit Administration Status	Used by the administrator to lock and unlock the object.
Plug-In Unit Availability Status	Provides further information regarding the state of the component.
Plug-In Unit Operative Status	Identifies whether or not the component is capable of performing its normal functions.
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.

Hardware Unit to Running Software Relationship

The SUN-SNMP-NETRA-CT-MIB hardware unit to running software relationship table is composed of the elements shown in TABLE 5-6.

TABLE 5-6 Hardware Unit to Running Software of SUN-SNMP-NETRA-CT-MIB

Field	Description
Hardware Running Table	Describes the software that is running on each hardware unit in the Netra CT NE.
Hardware Running Software Entry	An entry in the Netra CT Hardware Unit/Running Software relationship table. Each entry of this table identifies an entry in the entPhysicalTable and one in the hrSWRunTable.
Hardware Running Software Index	An index into the Netra CT Hardware Unit/Running Software relationship table.

Hardware Unit to Installed Software Relationship

The SUN-SNMP-NETRA-CT-MIB hardware unit to installed software relationship table is composed of the elements shown in TABLE 5-7.

TABLE 5-7 Hardware Unit to Installed Software of SUN-SNMP-NETRA-CT-MIB

Field	Description
Hardware Installed Software Table	Describes the software that is installed on each hardware unit in the Netra CT NE.
Hardware Installed Software Entry	An entry in the Netra CT Hardware Unit/Installed Software relationship table. Each entry of this table identifies an entry in the entPhysicalTable and one in the hrSWInstalledTable.
Hardware Installed Software to Software Index	The index, in the entPhysicalTable, of the containing physical entity in this pair.
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.

Alarm Severity Identifier Textual Convention

The SUN-SNMP-NETRA-CT-MIB alarm severity identifier textual conventions consist of the elements shown in TABLE 5-8.

TABLE 5-8 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, including 'cleared'.
Alarm Severity	The value of this object identifies the severity of an alarm that has occurred. (Note that there is no value corresponding to 'cleared'.)

Alarm Severity Profile

The SUN-SNMP-NETRA-CT-MIB alarm severity profile table consists of the elements shown in TABLE 5-9.

TABLE 5-9 Alarm Severity Profile Table of SUN-SNMP-NETRA-CT-MIB

Field	Description
Alarm Severity Default	The default severity value used for new profile index/trap ID pairs that have not yet been modified. This value is also used whenever an object's alarm severity profile index is set to 0. The default value of this object is minor(3).
Alarm Severity Profile Index Next	This object contains an appropriate value to be used for netraCtAlarmSevProfileIndex when creating entries in the netraCtAlarmSevProfileTable. The value -1 indicates that no unassigned entries are available. To obtain the index value for a new entry, the manager issues a management protocol retrieval operation to obtain the current value of this object. After each retrieval, the agent should modify the value to the next unassigned index (or -1).
Alarm Severity Profile Table	The Netra CT alarm severity profile table. This table specifies which profiles exist. Creating or deleting an entry in this table automatically creates or deletes the corresponding entries in the netraCtAlarmSeverityTable.

TABLE 5-9 Alarm Severity Profile Table of SUN-SNMP-NETRA-CT-MIB *(Continued)*

Field	Description
Alarm Severity Profile Entry	A group of severities, one for each alarm type in the communications alarm group.
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

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.)

TABLE 5-10 Alarm Severity Table of SUN-SNMP-NETRA-CT-MIB

Field	Description
Alarm Severity Entry	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.
Alarm Severity Trap ID	The ID of the trap type to which this entry applies.

Alarm Forwarding Discriminator

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.

TABLE 5-11 `SUN-SNMP-NETRA-CT-MIB` Alarm Forwarding Discriminator

Field	Description
Forward All Traps	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.
Trap Forwarding Table	The Netra CT Trap forwarding discriminator table specifies which traps will be sent to which management system.
Trap Forwarding Entry	Information about a group of traps to be sent to a particular IP address. Before its RowStatus column can be set to active(1), a new entry must have values for all attributes that do not have default values.
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

The SUN-SNMP-NETRA-CT-MIB trap agent MIB log table consists of the elements shown in TABLE 5-12.

TABLE 5-12 Trap Agent MIB Log Table of SUN-SNMP-NETRA-CT-MIB

Field	Description
Trap 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.
Trap Log Entry	Information about a single trap log.
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.
Trap Log Administrative Status	The management system uses this object to stop and start the operations of this object.
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.
Trap Log Row Status	This object is used to create a new row or to delete an existing row in the table.

Trap Agent MIB Logged Trap

The SUN-SNMP-NETRA-CT-MIB Trap Agent Logged Trap table is used to maintain the traps logged and consists of the elements shown in TABLE 5-13.

TABLE 5-13 Trap Agent MIB Logged Trap Table of SUN-SNMP-NETRA-CT-MIB

Field	Description
Logged Trap Entry	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.
Logged Trap Index	A unique number identifying this entry in the log. When the maximum value for this object has been reached, it wraps around to 0.
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).

Trap Agent MIB Logged Alarm

The SUN-SNMP-NETRA-CT-MIB Trap Agent MIB Logged Alarm table consists of the elements shown in TABLE 5-14.

TABLE 5-14 Trap Agent MIB Logged Alarm of SUN-SNMP-NETRA-CT-MIB

Field	Description
Logged Alarm Table	The Netra CT Trap Agent logged alarm trap table is used to maintain extra information for logged traps that represent alarm types.
Logged Alarm Entry	Information about the alarm-specific attributes of a single trap in the log.
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.
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 5-15 can be optionally appended to any alarm notification.

TABLE 5-15 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.

TABLE 5-16 MIB Notifications

Field	Description
Hardware Unit High Temperature Alarm	Indicates that a high temperature condition has occurred on the hardware unit associated with the specified index. An `entPhysicalClass` of unknown(2) along with both an `entPhysicalContainedIn` of 0 and an `entPhysicalParentRelPos` of -1 indicates that the error occurred in the Netra CT NE but not in any one hardware unit maintained in the MIB table.

State Change Notification Traps

The SUN-SNMP-NETRA-CT-MIB State Change Notification Traps table consists of the elements shown in TABLE 5-17.

TABLE 5-17 State Change Notification Traps of SUN-SNMP-NETRA-CT-MIB

Field	Description
Hardware Unit Up	Indicates that the operational state of the specified hardware unit has changed to up.
Hardware Unit Down	Indicates that the operational state of the specified hardware unit has changed 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 Object Creation and Deletion Notification Traps of SUN-SNMP-NETRA-CT-MIB

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	(Not supported on Netra CT 820.) Indicates that the specified software package has been installed.
Installed Software Deleted	(Not supported on Netra CT 820.) 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 Configuration Change Notification Traps of SUN-SNMP-NETRA-CT-MIB

Field	Description
Interface Changed	(Not supported on Netra CT 820.) 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	(Not supported on Netra CT 820.) 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. Can be one of 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, 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 can be used at the discretion of an SNMP agent when an object instance is created.

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 DMC 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.

▼ To Change the Midplane FRU-ID

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 result 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.1 = STRING: midplane
ENTITY-MIB::entPhysicalDescr.2 = STRING: ps
ENTITY-MIB::entPhysicalDescr.3 = STRING: fan_tray_slot
ENTITY-MIB::entPhysicalDescr.4 = STRING: fan
ENTITY-MIB::entPhysicalDescr.6 = STRING: cpsb_slot
ENTITY-MIB::entPhysicalDescr.7 = STRING: cpsb_slot
     :                :            :        :
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**.

The Netra CT DMC 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:9161 \
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 hostName: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 hostName: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 hostName: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 hostName: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 hostName: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 hostName: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` MIB.

CHAPTER 6

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 distributed management cards (DMCs) and all of the node CPU cards in a single Netra CT system, or it can include a defined group of DMCs and node CPU cards located in multiple systems.

The PMS software provides distributed node card resource management infrastructure for clusters of node cards. This infrastructure includes low-level administrative control and monitoring, high-level configuration, fault recovery, and user-interface functionality. FIGURE 6-1 identifies the architectural components of the Netra CT software services.

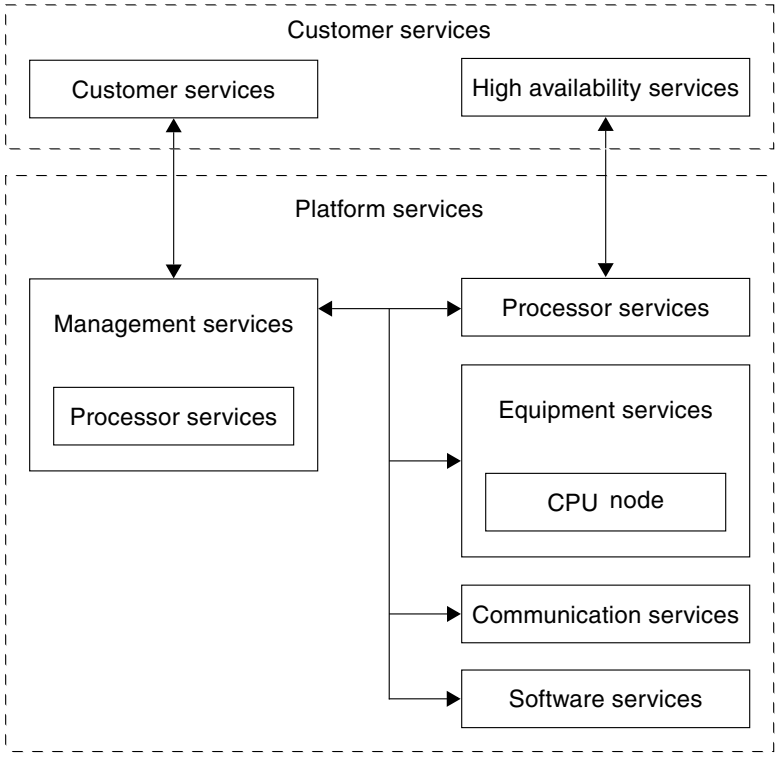


FIGURE 6-1 Netra CT Software Services

In a Netra CT cluster, the PMS software runs on both the DMCs and the node cards. The PMS software running on DMCs provides local and remote service connections for managing the node cards in its system. The PMS software running on node 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 node cards in a PMS cluster.

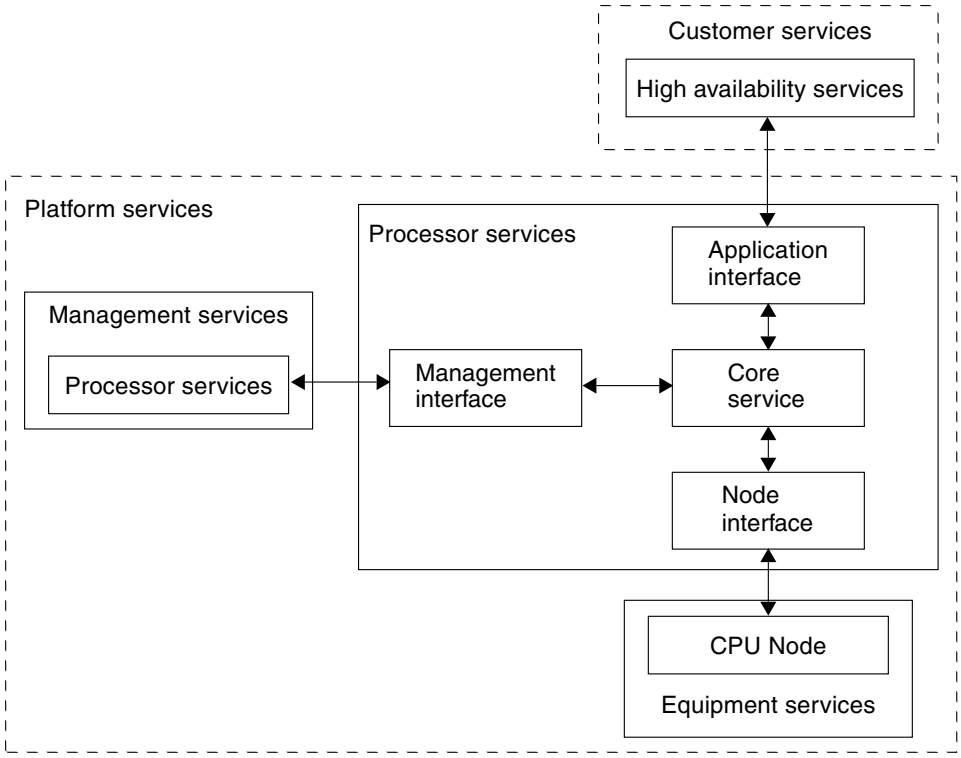


FIGURE 6-2 PMS Software Services and Interfaces

FIGURE 6-2 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 DMCs and the node 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 DMC 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 node cards are divided into *management, node* and *remote node drawer* (RND) views. The management view on both the DMC and the node card 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 DMC 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, operational, and non-configuration, five recovery processes, and the graceful reboot of the group.

The node view, by way of the node 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 node card RND view provides remote system view administrative controls and monitors to all the resource groups, with the exception of a distributed management 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 node card growth is addressed by a per-CPU multithreading of up to 16 remote node 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 6-1 lists the man pages included with the Netra CT PMS software:

TABLE 6-1 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 node card PMS daemon (`pmsd`) and lists the daemon's command-line options.
pmsd_ac(1M)	Describes how to start and stop the DMC PMS daemon (`pmsd_ac`) from the command-line interface, and lists all the daemon's other command-line functions.
pms_apistart(1M) pms_start(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.
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.

TABLE 6-1 Processor Management Services Man Pages *(Continued)*

Man page	Description
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/mgmt3.0/man`. Depending on the UNIX shell you are using, this variable can 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 66
- "PMS Client Main Thread" on page 73
- "PMS Client Asynchronous Message Handling" on page 75
- "PMS Client Scheduling" on page 88
- "PMS Client User and Management Interface" on page 89
- "PMS Client Node Interface" on page 106
- "PMS Client RND Interface" on page 113

CODE EXAMPLE 6-1 begins by initializing the main thread for a PMS client.

CODE EXAMPLE 6-1 PMS Client Initialization Example

```
#include <sys/types.h>          /* socketpair() */
#include <sys/socket.h>         /* socketpair() */

#include <unistd.h>             /* write(), read() */

#include <signal.h>             /* sigemptyset(), sigaddset(), sigaction() */
#include <time.h>               /* timer_create(), timer_settime() */

#include <stdio.h>              /* printf(), scanf() */
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
#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);

/* 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);
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
/* 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
                {
#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
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
                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;
                int                                 last_update;
                int                                 last_sync_check;

                int                                 rg0_state_cache;

                } node_info;

        struct
                {
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
#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");

    exit(1);
    }

  /* Start/Initialize the PMS API before using any further calls.. */

  if (pms_apistart() == -1)
    exit(2);
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
/* 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;

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;
  }
```

CODE EXAMPLE 6-1 PMS Client Initialization Example *(Continued)*

```
  /* 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 6-2 shows the main thread.

CODE EXAMPLE 6-2 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);

        if (receivestatus <= 0)
          {
```

CODE EXAMPLE 6-2 PMS Client Main Thread *(Continued)*

```
          /* 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;

  app_hasim_receive_post(&pr);

}
```

CODE EXAMPLE 6-3 sets up a PMS client to handle asynchronous messages.

CODE EXAMPLE 6-3 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);

    break;

    case PMS_PD_NODE_RG0_STATUS:

      app_hasim_receive_node_rg0_status(pr);

    break;

    case PMS_PD_NODE_RG0_APP_STATE_SET_EXECUTE:
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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");


      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:
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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:

      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");
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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:

     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");
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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;
            }
          }
        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];
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
          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;
    }

}

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");
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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:

      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;

    }

}
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
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;
      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;
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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)
{


  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");
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      /* 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;
    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:
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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", \
     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");
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      /* 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.. */

    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:
```

CODE EXAMPLE 6-3 PMS Client Asynchronous Message Handling *(Continued)*

```
      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");

      /* Doing nothing at the moment.. */

    break;
    }
}
```

CODE EXAMPLE 6-4 shows a PMS client's scheduling.

CODE EXAMPLE 6-4 PMS Client Scheduling

```
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.. */

  app_hasim_user_process();

  app_hasim_mgmt_process();

  app_hasim_node_process();

  app_hasim_rnd_process();


}
```

CODE EXAMPLE 6-5 shows the PMS client's user management interface.

CODE EXAMPLE 6-5 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;
      }
    }
  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];
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
            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;
          }

        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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
    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)
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
        {
        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;
      }
    }

}

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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
/* 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;
      }
    }
  }


/* 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)
      {
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
        /* 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;
            }

          }


          /* 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;
          }
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
      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;
        }


        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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
          /* 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], \
                       &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)
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
                    {
                    }
                  }
                }

              }
            }

          /* 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;

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
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
     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;
          }
        }

      }
    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;
```

CODE EXAMPLE 6-5 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;
            }

          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)
              {
              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;
            }
          }

        }

      }

    }
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
/* 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)
        {
        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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
            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], \
&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);
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
                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;
     }
   }


/* 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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
 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;
  }

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);
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
        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;

      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;
```

CODE EXAMPLE 6-5 PMS Client User and Management Interface *(Continued)*

```
        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;
        }

      }
    }


}
```

CODE EXAMPLE 6-6 shows the PMS client node interface.

CODE EXAMPLE 6-6 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 6-6 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 6-6 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 6-6 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;

            if (pms_send(&ps, &pr) == 0)
              {
              if (pr.payload.data.node_rg0_info_get_status.err == \
```

CODE EXAMPLE 6-6 PMS Client Node Interface *(Continued)*

```
                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
     initial configuration.. */

  if(mdi.node_info.last_update < HASIM_CHECK_VALID_INTERVAL)
    {
    if (mdi.node_info.cache_state != HASIM_NODE_CACHE_VALID)
```

CODE EXAMPLE 6-6 PMS Client Node Interface *(Continued)*

```
      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;
          }

        }

      /* NODE*/

      if (mdi.node_info.service_state == HASIM_NODE_SERVICE_STATE_ACTIVE)
        {
        mdi.node_info.service_state = HASIM_NODE_SERVICE_STATE_OFFLINE;
        }
```

CODE EXAMPLE 6-6 PMS Client Node 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;
          }

        }
      }


}
```

CODE EXAMPLE 6-7 shows a PMS client RND interface.

CODE EXAMPLE 6-7 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 6-7 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 6-7 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 6-7 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))
      {
      if (mdi.rnd_info[i].cache_state == HASIM_RND_CACHE_VALID)
```

CODE EXAMPLE 6-7 PMS Client RND Interface *(Continued)*

```
        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 7

Solaris Operating System APIs

This chapter introduces Solaris operating system APIs of concern to the Netra CT 820 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 – 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.

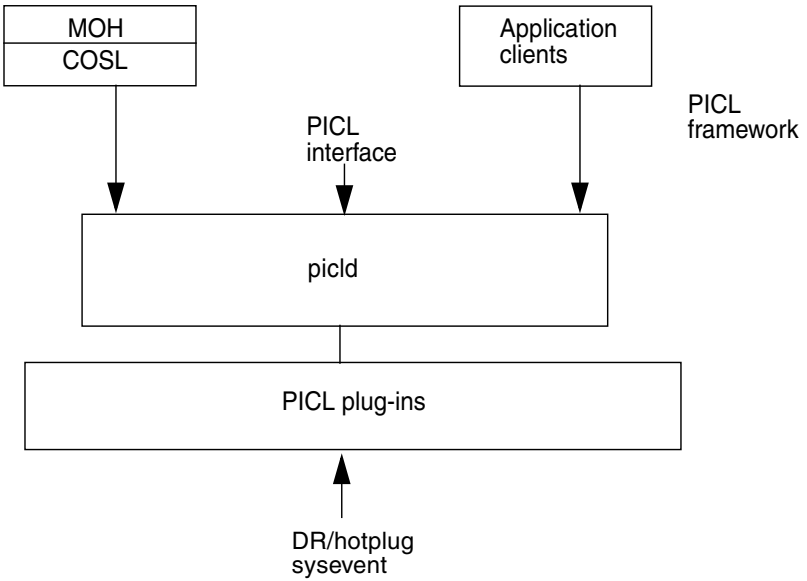


FIGURE 7-1 PICL Daemon (`picld`) and Plug-ins

FIGURE 7-1 diagrams the PICL daemon (`picld`) and its connection to the PICL plug-ins, some of which are common to all platforms. The DR/hotplug PICL `sysevent` notifies the `piclevent` plug-in through a private interface.

Application clients use `libpicl`(3LIB) to interface with `picld`. They "see" and 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.

Communication between nodes is enabled by the Ethernet interface over the cPSB in accordance with PICMG 2.16. The interface is configured automatically during system startup.

The following section identifies the exported interfaces to the frutree plugin.

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 7-1. Each of these classes is populated with various properties, among which are State and Condition. Details follow the summary table.

TABLE 7-1 PICL FRUtree Topology

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.
`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.

TABLE 7-1 PICL FRUtree Topology *(Continued)*

Node Class	Properties	Description
	StatusTime	Time when State was updated last.
	Condition	Condition of the port: ok, unknown, failing, failed, or testing.
	ConditionTime	Time when Condition was updated last.
	Devices	Table of node handles in platform 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
```

Fru Class Properties

Where the following **fru** class properties are writable, 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 of node represents the **occupant** state of the attachment point associated with the fru node. State value properties are `configured unconfigured`, or `unknown.`

Fru Condition

`CHARSTRING` read-only

The **Condition** property of the fru class of node represents the **occupant** condition of the attachment point associated with the fru node. Condition values are shown in TABLE 7-2.

TABLE 7-2 PICL FRU Condition Value Properties

FRU Condition	Description
`unknown`	FRU condition could not be determined.
`ok`	FRU is functioning as expected.
`failing`	A failure has been predicted.
`failed`	FRU has failed.
`unusable`	FRU is unusable for undetermined reason.

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 may be dependencies, such as when a remote device is cabled to a port node. These dependencies may 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 7-3:

TABLE 7-3 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.

Condition

`CHARSTRING` read-write

The **Condition** value of a port class node carries the meaning shown in TABLE 7-4.

TABLE 7-4 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 7-5.

TABLE 7-5 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 820 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. 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 7-6.

TABLE 7-6 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 7-7 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 7-7 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.
`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.

TABLE 7-7 PICL Man Pages *(Continued)*

Man Page	Description
`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 these functions, see "Setting the Watchdog Timer Properties Using the PICL API" on page 130.

Reading Temperature Sensor States Using the PICL API

Temperature sensor states can be read using the `libpicl` API. TABLE 7-8 lists the properties that are supported in a PICL temperature sensor class node.

TABLE 7-8 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.

Threshold levels of the PICL node class `temperature-sensor` are:

- Warning
- Shutdown
- PowerOff

TABLE 7-9 lists the PICL threshold levels and their MOH equivalents.

TABLE 7-9 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 820 server is shown in CODE EXAMPLE 7-1.

CODE EXAMPLE 7-1 Example Output of PICL Temperature Sensors

```
# prtpicl -c temperature-sensor -v
 CPU-sensor (temperature-sensor, 450000039e)
            :State          ok
            :LowPowerOffThreshold  -20
            :HighWarningThreshold  74
            :HighShutdownThreshold          79
            :HighPowerOffThreshold          91
            :LowWarningThreshold   -10
            :LowShutdownThreshold  -13
            :Temperature            59
            :GeoAddr        0xe
            :Label          Ambient
            :_class         temperature-sensor
            :name           CPU-sensor
```

Setting the Watchdog Timer Properties Using the PICL API

The Netra CT 820 server watchdog service captures catastrophic faults in the Solaris OS running on the node card. The watchdog service reports such faults to the DMC by means of either an IPMI message or by a de-assertion of the CPU's HEALTHY# signal.

The Netra CT 820 server 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 node card on which the watchdog timer is to be changed.

   To do this, switch to the DMC CLI and use the command `pmsd infoshow`, specifying the slot number of the node card. 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 node 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 7-10, TABLE 7-11, and TABLE 7-12.

3. Use the PMS API to enable the primary HEALTHY# signal monitoring on the CPU card in the specified slot.

   From the DMC 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 6, Processor Management Services, for information on PMS.

PICL interfaces for the watchdog plug-in module (see TABLE 7-10) include the nodes watchdog-controller and watchdog-timer.

TABLE 7-10 Watchdog Plug-in Interfaces for Netra CT 820 Server Software

PICL Class	Property	Meaning
`watchdog-controller`	`WdOp`	Represents a watchdog subsystem.
`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 7-11 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 7-12 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 startup 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#.

TABLE 7-12 Properties Under `watchdog-timer` Node *(Continued)*

Property	Values	Description
	`reset`	Perform a soft or hard reset of the system (implementation specific).
	`reboot`	Reboot the system.

* A platform might not support a specified timeout resolution. For example, Netra CT 820 servers only take -1, 0, and 100 6553500 ms in increments of 100 msec. (Level 1), and -1, 0, 255000 ms in increments of 1000 msec.(Level 2).

† A specific timer node might not support all action types. For example, Netra CT 820 server watchdog level 1 timer supports only `none`, `alarm`, and `reboot` 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 7-2.

CODE EXAMPLE 7-2 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 cPSB 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. CODE EXAMPLE 7-3 shows an example of the output of the prtfru command.

CODE EXAMPLE 7-3 Sample Output of `prtfru` Command

```
# prtfru
/frutree
/frutree/chassis (fru)
/frutree/chassis/CPU?Label=CPU
/frutree/chassis/CPU?Label=CPU/CPU (container)
   SEGMENT: SD
      /ManR
      /ManR/UNIX_Timestamp32: Thu Jun 19 19:47:57 PDT 2003
      /ManR/Fru_Description: ASSY,CPCI,CP2300,SNOWBIRD
      /ManR/Manufacture_Loc: MITAC, TAIWAN
      /ManR/Sun_Part_No: 3753121
      /ManR/Sun_Serial_No: 002641
      /ManR/Vendor_Name: Sun Microsystems
      /ManR/Initial_HW_Dash_Level: 07
      /ManR/Initial_HW_Rev_Level: 07
      /ManR/Fru_Shortname: CPU
      /SpecPartNo: 885-0110-07
/frutree/chassis/CPU?Label=CPU/CPU/PMC-1?Label=PMC-A
/frutree/chassis/CPU?Label=CPU/CPU/PMC-1?Label=PMC-A/PMC-1 (fru)
/frutree/chassis/CPU?Label=CPU/CPU/PMC-1?Label=PMC-A/PMC-1/c1::rmt/0?Label=4
/frutree/chassis/CPU?Label=CPU/CPU/PMC-1?Label=PMC-A/PMC-1/c1::rmt/0?Label=
4/c1::rmt/0 (fru)
/frutree/chassis/CPU?Label=CPU/CPU/PMC-2?Label=PMC-B
/frutree/chassis/CPU?Label=CPU/CPU/PMC-2?Label=PMC-B/PMC-2 (fru)
/frutree/chassis/CPU?Label=CPU/CPU/PMC-2?Label=PMC-B/PMC-2/disk1?Label=0,0
/frutree/chassis/CPU?Label=CPU/CPU/PMC-2?Label=PMC-B/PMC-2/disk1?Label=
0,0/disk1 (fru)
/frutree/chassis/RTM?Label=RTM
/frutree/chassis/RTM?Label=RTM/RTM (fru)
/frutree/chassis/RTM?Label=RTM/RTM/CDROM1?Label=1,0
/frutree/chassis/RTM?Label=RTM/RTM/CDROM1?Label=1,0/CDROM1 (fru)
#
```

Glossary

A

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.

Agent Application Program written in the Java language containing an MBean server that can manage resources and services.

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 A value of any type that a manager application can get or set remotely.

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 DMC.

D

DMC Distributed management card. The DMC is used in the Netra CT 820 server to provide system control functions. The DMC resides in slot 1A and is paired with another in slot 1B to provide failover redundancy in the Netra CT 820 server.

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, enables the management system to control the behavior of the log.

G

GPIO General purpose I/O.

I

IM Information model.

IPMI Intelligent Platform Management Interface, used as a communication channel over the compactPCI 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

MBean Managed Bean, a Java object that represents a resource's managed object interface. The object follows design patterns set up for the MBean interface.

MIB Managed information base used to describe the exchange of information across the network element (NE) interface. A MIB is loadable, but may 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 DMC and any node 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.

Node Card CPU card residing in slots 3-20 in the Netra CT 820 system.

O

Operation A method with any signature and any return type that the manager application can invoke remotely

P

Physical Path Termination Point See Termination Point MBean.

PICL Platform Information and Control Library. A Solaris OS library that provides a method 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.

Resource Any entity, physical or virtual, to be monitored through the network.

RG0 Resource Group 0 (Applications)

RG1 Resource Group 1 (Operating System)

RG2 Resource Group 2 (Hardware)

RMI Remote Method Invocation. Java RMI is a mechanism that enables you to invoke a method on an object that exists in another address space.

RNFS Reliable Network File System.

S

SMI Structure of Management Information. A definition that describes the syntax and basic data types available in a given MIB.

SNMP Simple Network Management Protocol. A protocol that enables devices to be controlled remotely by a network management station.

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

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.

TFTP Trivial File Transfer Protocol.

Topology Change Notification An abstract class representing generic notifications for a change in the topology of a network entity.

Trap Unsolicited network packet sent from an agent that usually reports some error condition.

Index

E

F

G

H

I

J

M

N

O

P

R

S

T

V

W