Netra High Availability Suite Foundation Services 2.1 6/03 NMA Programming Guide

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

Part No: 817–1771–11
September 2004

040812@9495

Contents

Examples

Figures

Preface

This book describes how to use the Node Management Agent (NMA) Java™ APIs of the Netra™ High Availability (HA) Suite Foundation Services 2.1 6/03. This book can be used to perform the following tasks:

- Access cluster information from a remote Java or Simple Network Management Protocol (SNMP) management and monitoring application
- Receive SNMP traps or JMX™ notifications about changes occurring in the cluster
- Provoke a cluster mastership switchover
- Manipulate Daemon Monitor parameters

Who Should Use This Book

This book is for application developers who want to develop applications that use the NMA.

Before You Read This Book

To program the NMA you must have working knowledge of the Java language. Knowledge of the Java Dynamic Management Kit (DMK) 5.0 and the Solaris™ operating system is an advantage.

Before reading this book, read the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

How This Book Is Organized

This book contains the following chapters:

- Chapter 1 outlines the features of the NMA.
- Chapter 2 introduces and defines the requirements and dependencies of the NMA.
- Chapter 3 describes how to access the NMA from a Java manager.
- Chapter 4 describes how to access the NMA from a SNMP manager.
- Chapter 5 explains how to use the NMA to provoke a master node switchover and set Daemon Monitor parameters.
- Chapter 6 describes the Carrier Grade Transfer Protocol (CGTP) statistics that can be accessed from the NMA.
- Chapter 7 describes the Daemon Monitor statistics that can be accessed from the NMA.
- Chapter 8 describes the Reliable NFS statistics that can be accessed from the NMA.
- Chapter 9 describes the Cluster Membership Manager (CMM) statistics that can be accessed from the NMA.
- Chapter 10 explains the NMA notification mechanism and the meaning of the NMA notification types.
- Appendix A describes the syntax of the MBean naming conventions.

Chapter 4 refers to RFC standards. For further information, see the complete text of RFC papers at `http://www.ietf.org/`.

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

Related Books

You will require some of the following books from the Foundation Services documentation set:

- *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*

- *Netra High Availability Suite Foundation Services 2.1 6/03 Glossary*
- *What's New in Netra High Availability Suite Foundation Services 2.1 6/03*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Quick Start Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Hardware Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Custom Installation Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Cluster Administration Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Troubleshooting Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 CMM Programming Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 NMA Programming Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Reference Manual*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Standalone CGTP Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Release Notes*
- *Netra High Availability Suite Foundation Services 2.1 6/03 README*

Java DMK documentation can be found on the Internet site http://www.sun.com/products-n-solutions/nep/software/java-dynamic/.

- *Getting Started with the Java Dynamic Management Kit 5.0*
- *Java Dynamic Management Kit 5.0 Tutorial*

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print form. For a list of documents and how to order them, see "Buy printed documentation" at `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. (Emphasis sometimes appears in bold online.)

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

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

TABLE P–2 Shell Prompts *(Continued)*

Shell	Prompt
Bourne shell and Korn shell superuser prompt	#

CHAPTER 1

Introduction to the Node Management Agent

This chapter describes how to access the Node Management Agent (NMA) and introduces the master node view and the floating external address.

The NMA is a management agent that conforms to the Java Management Extensions (JMX) v1.1 Maintenance Release. The NMA is based on the Java Dynamic Management Kit (DMK) 5.0 APIs.

The NMA monitors the performance and the status of the following Foundation Services:

- Cluster Membership Manager
- Carrier Grade Transport Protocol (CGTP)
- Daemon Monitor
- Reliable NFS

The NMA exposes this information through a Simple Network Management Protocol (SNMP) management information base (MIB) and a JMX compliant interface. The JMX specification defines a three-level management architecture:

- The instrumentation level makes resources manageable as Java objects called MBeans.
- The agent level exposes these objects for management.
- The distributed services level allow remote access and security.

By programmatically accessing these MBeans, management applications can be used to help perform tuning operations, diagnostic operations, and troubleshooting.

The NMA also provides a method for provoking a master node switchover, and emits notifications which can be used to keep up-to-date with the current state of certain aspects of the cluster.

The Java DMK implements the JMX specification. The NMA requires the Java DMK runtime libraries. You can use the Java DMK to develop a remote Java manager to access the NMA. Alternately, access the NMA MIB by using an off-the-shelf or custom SNMP manager, or any JMX compliant Java manager.

This chapter contains the following sections:

Accessing the NMA

An external manager can communicate with the NMA using any of the following protocols:

- HTTP
- SNMP version 1 (SNMPv1)
- SNMP version 2 (SNMPv2)
- SNMP version 3 (SNMPv3)

Note – The floating external address cannot be used when you are using the SNMP protocol. See "Floating External Address" on page 21 for more information on the floating external address.

Use SNMPv3 to take advantage of the enhanced security mechanism introduced in SNMPv3. Note that not all protocol interfaces are enabled by default. See `nma.properties`(4) for information on how to enable, disable and configure protocol interfaces.

HTML can be used to view the NMAs in a running cluster through an HTML browser. By default port `8082` exports this view. To interact with the NMA on a node, access the URL of the form `http://`*nodeIPAddress*`:`*portNumber*

See "HTML Protocol Adaptor" in the *Java Dynamic Management Kit 5.0 Tutorial* for more information.

Note that if you provoke a switchover using the HTML Protocol Adaptor connected to the floating external address, the connection to the NMA might be broken prematurely and the information transfer will not finish. If this happens, stop the transfer and reload the page to get the correct node information.

Figure 1–1 represents the communication paths between an external manager and the cluster:

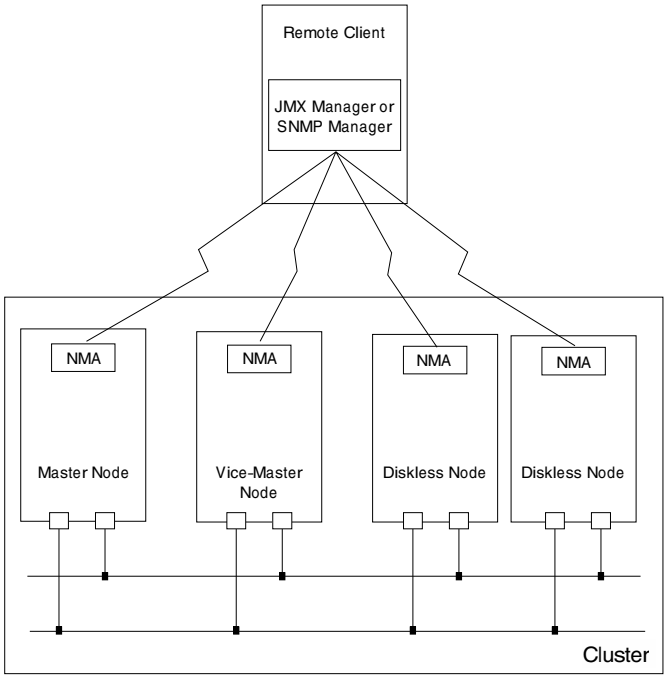

FIGURE 1–1 Remote Manager Communication

Master and Node Views

The NMA running on the master node makes information from the NMAs running on all nodes visible in the NMA running on the master node through a cascading connection. The MBeans listed in "MBean Instances on the Master Node" on page 21 are available on the master node only.

Figure 1–2 shows how information is cascaded from all NMAs to the NMA on the master node.

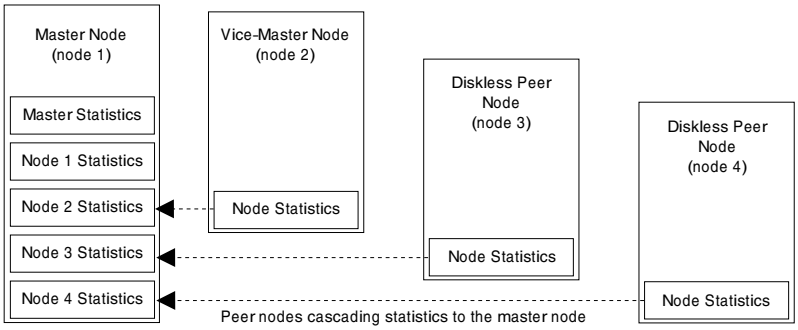

FIGURE 1–2 Cascading Information From Peer Nodes to the Master Node

---

Note – The master view is only available to a Java manager that communicates using the HTTP protocol.

All of the agents must use the same port number for the service used to implement cascading. If this is not the case, the master agent will start the cascading service but will not enable the cascading connections to NMA.

Note – After failover or switchover, there is a short period of time during which the NMA information of other nodes is made available on the new master. If you query this information from the master during this period, an exception will be thrown.

Cascading is controlled by the following properties, which are defined in the `nma.properties` file:

`com.sun.nhas.ma.cascading.enabled`
Set to `true` to enable cascading.

`com.sun.nhas.ma.cascading.retries.max`
The maximum number of times that the master node tries to create a cascading connection to an NMA on another node.

`com.sun.nhas.ma.cascading.retries.delay`
The time in milliseconds between each attempt to establish a cascading connection.

`com.sun.nhas.ma.cascading.comm.protocol`
The protocol can be `http` or `rmi`.

`com.sun.nhas.ma.cascading.socket.timeout.wait`
The timeout in milliseconds for a cascading connection to terminate cleanly in the case of a communication fault. After this time has elapsed, the cascading connection attempt will be forcibly aborted and then restarted.

MBean Instances on the Master Node

One instance of each of the following MBeans is instantiated on the master node of the cluster.

- `CmmMasterNodeMBean`
- `CmmMasterStatisticsMBean`
- `RnfsMasterStatisticsMBean`
- `PmdMasterStatisticsMBean`
- `CgtpMasterMBean`

An `RnfsReplicatedSliceMBean` is instantiated on the master node and on the vice-master node for each Reliable NFS partition.

MBean Instances on Peer Nodes

One instance of each of the following MBeans is instantiated on each peer node of the cluster:

- `ClusterNodeMBean`
- `CmmStatisticsMBean`
- `RnfsStatisticsMBean`
- `PmdStatisticsMBean`
- `CgtpMBean`
- `CgtpFilterMBean`

Multiple instances of the following MBeans might be instantiated:

`CgtpReliableLinkStatisticsMBean`	One instance for each reliable link.
`PmdNameTagStatisticsMBean`	One instance for each group of processes monitored by the Daemon Monitor.

Floating External Address

An external Java manager can use the floating external address (for example, `10.250.10.1`) to communicate with the master node. After failover and switchover the floating address is reassigned to the new master node. The external Java manager must then connect to the new master node, but can connect to the same IP address as before. SNMP cannot use the floating external address to communicate with the master node.

CHAPTER 2

Configuration Files, Dependencies and Requirements

This chapter defines the NMA software requirements and dependencies.

This chapter contains the following sections:

Configuration Files

The following table summarizes the NMA configuration files.

TABLE 2–1 NMA Configuration Files

File	Description
`/etc/opt/SUNWcgha/nma.properties`	NMA properties file. See `nma.properties`(4) for more information.
`/etc/opt/SUNWcgha/nma.security`	NMA SNMPv3 security parameter configuration file. See `nma.security`(4) for more information.
`/etc/opt/SUNWcgha/nma.notifs.txt`	SNMP trap identification configuration file. See Chapter 4 for more information.
`/etc/opt/SUNWcgha/nma.params.txt`	SNMP trap parameter configuration file. See Chapter 4 for more information.
`/etc/opt/SUNWcgha/nma.targets.txt`	SNMP trap target configuration file. See Chapter 4 for more information.

TABLE 2–1 NMA Configuration Files *(Continued)*

File	Description
`/etc/opt/SUNWcgha/nma.uacl.template`	Template for SNMPv3 user access configuration file. See Chapter 4 for more information.
`/etc/opt/SUNWcgha/nma.acl.template`	Template for SNMPv1, SNMPv2 and SNMPv3 IP configuration file. See Chapter 4 for more information.

SNMP applications can also manipulate SNMPv3 configuration by using the `com.sun.jdmk.snmp.rfc2573.managerSnmpV3AppliMibRegistration` class.

Dependencies

To enable an external Java manager to access NMA statistics, the class path of the external Java manager must contain the following Java Archive (JAR) files:

- *installDir*`/SUNWcgha/lib/jcmm.jar`
- *installDir*`/SUNWcgha/lib/ma.jar`
- *installDir*`/SUNWcgha/lib/cghautil.jar`
- *installDir*`/SUNWjdmk/jdmk5.0/lib/jdmkrt.jar`
- `/usr/sadm/lib/snmp/jsnmpapi.jar`
- *installDir*`/SUNWcgha/lib/rfc2573mgr.jar`
- *installDir*`/SUNWcgha/lib/rfc2573.jar`

These JAR files are contained in the `SUNWnhmaj` package, the `SUNWjdrt` package, and the `SUNWjsnmp` package.

Software Requirements

The NMA requires the following versions of Solaris:

- Java 2 Runtime Environment v1.3.1 for Solaris 8 2/02 and Solaris 8 PSR 1
- Java 2 Runtime Environment v1.4 for Solaris 9 and Solaris 9 9/02

A Java DMK client requires the Java DMK 5.0 runtime libraries, but the usage of Java DMK communication interfaces is configurable. Both Java DMK 4.2 clients and Java DMK 5.0 clients are supported. To interface with Java DMK 4.2 clients, set the value of the `jmx.serial.form` property to `1.0`

CHAPTER 3

Developing an External Java Manager

For information about how to develop an external Java manager, see the following sections:

Configuring an External Java Manager Using HTTP

To use the HTTP protocol adaptor, edit the following NMA properties in the `nma.properties` file:

`com.sun.nhas.ma.connectors.http.enabled`
Set to `true` to enable the HTTP protocol adaptor

`com.sun.nhas.ma.connectors.http.port`
Set to the number of the port to be used for HTTP communication, for example, `8081`

These properties are by default `true` and `8081` respectively.

Connecting to the NMA

The procedure for connecting to the NMA, or reconnecting to the NMA in case of a change of mastership, depends on your addressing scheme.

Using the Floating Address

If you are using the floating address to connect to the NMA running on the master node, perform the following steps to manage a failover or switchover:

1. Use the Java DMK heartbeat mechanism to detect the loss of contact with the master node.

   Reduce the timeouts on requests if necessary in order to guarantee the timely detection of a master node crashing abruptly. See "Heartbeat Mechanism" in the *Java Dynamic Management Kit 5.0 Tutorial* for information about how to use the Java DMK heartbeat mechanism.

2. Reconnect to the NMA.

   The master floating address will be assigned to the new master node of the cluster. Reconnect to this NMA at this address.

3. Wait until the cascading service has finished restarting and the master view has restarted.

   The cascading service queries all of the NMAs running in the cluster and makes this information available from the NMA on the master node. During this service restart period, not all MBeans cascading from other nodes will be available. If you attempt to manipulate an MBean on the master node that has not yet cascaded, an `InstanceNotFound` exception is thrown.

4. Reregister all notification listeners.

   Notifications might be lost between when mastership changes and listeners are reregistered. Query the NMAs in the cluster to discover the current cluster state.

Using a Physical Node Address

If you are connecting to the NMA on each node using the node's IP address no connections will fail after failover or switchover.

Using Proxy MBeans

The statistics providers in the NMA are implemented as MBeans. A set of generated proxy classes for the statistics MBeans is supplied with the NMA. A remote manager can access these statistics through the exposed MBean interfaces. The Java DMK enables predefined proxy classes of these MBeans to be instantiated in an external Java manager, and the objects to be manipulated as if they were present locally. Communication with the proxied MBeans is handled automatically.

For more information about using proxy MBeans, see "MBean Proxies" in the *Java Dynamic Management Kit 5.0 Tutorial*.

To use the supplied proxy classes, the Java DMK Remote Manager's class path must contain the path to `proxies_42.jar` or `proxies_50.jar`, depending on the version of the Java DMK runtime that you are using. The Java DMK toolkit can be used to regenerate the proxy classes. See Appendix A for more information.

CHAPTER 4

Developing a Remote SNMP Manager

NMA information can be accessed using the Simple Network Management Protocol (SNMP). This chapter explains how to configure an external SNMP manager, and provides examples of the configuration files required for three types of SNMP configurations.

The Java DMK can be used to develop a remote manager that communicates with the NMA using SNMP. For information on how to use the Java DMK to develop a manager that communicates using SNMP, see "Developing an SNMP Manager" in the *Java Dynamic Management Kit 5.0 Tutorial*. Alternately, any SNMP manager can be used.

This chapter contains the following sections:

Configuring an SNMP Agent

The NMA offers SNMPv1, SNMPv2 and SNMPv3 interfaces via the SNMP protocol adaptor. Edit the following values in the `nma.properties` file to configure the SNMP protocol adaptor:

`com.sun.nhas.ma.adaptors.snmp.enabled`
Set to `true` to enable the SNMP protocol adaptor

`com.sun.nhas.ma.adaptors.snmp.port`
Set to number of the port to be used for SNMP communication, for example, `8085`

`com.sun.nhas.ma.adaptors.snmp.trap.port`
Set to number of the port to be used to send SNMP traps, for example, `8086`

By default the NMA uses the standard Java DMK access control configuration files. The following templates are available for use in a default installation:

installDir`/etc/opt/SUNWcgha/nma.acl.template`
Used for SNMPv1, SNMPv2 and SNMPv3 IP access. For SNMPv3, IP access is only relevant to SNMP traps.

installDir`/etc/opt/SUNWcgha/nma.uacl.template`
Used for SNMPv3 user access only.

Use these templates to create configuration files for customized access control configuration. Edit the `jdmk.acl.file` and `jdmk.uacl.file` properties in the `nma.properties` file to reflect the paths to your access control configuration files.

The following three files are included for SNMP traps and security configuration, in accordance with the Internet Engineering Task Force RFC 2573.

installDir`/etc/opt/SUNWcgha/nma.targets.txt`
SNMP trap target configuration file.

installDir`/etc/opt/SUNWcgha/nma.params.txt`
SNMP trap security parameter configuration file.

installDir`/etc/opt/SUNWcgha/nma.notifs.txt`
SNMP trap identification configuration file.

The NMA MIB is located at `/opt/SUNWcgha/etc/ma/nhasmib.txt` in a default installation.

Note – SNMPv1 does not support 64–bit counters. Retrieval of CGTP statistics that use 64–bit counters is not possible when using SNMPv1.

IP-Based Access Control Lists

In SNMPv1 and SNMPv2, access control is provided on the basis of the IP address and community of the manager's host machine.

EXAMPLE 4–1 Typical `nma.acl` File

```
acl = {
 {
 communities = public
 access = read-only
 managers = yourmanager
 }
 {
```

EXAMPLE 4–1 Typical `nma.acl` File *(Continued)*

```
 communities = private
 access = read-write
 managers = yourmanager
 }
}

trap = {
  {
  trap-community = public
  hosts = yourmanager
  }
}
```

Format of the `acl` Group

The `acl` group contains one or more access configurations.

```
acl = {
   access1
   access2
      ...
   accessN
}
```

Each access configuration has the following format:

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

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

The *accessRights* specifies the rights to be granted to all managers connecting from the hosts specified in the *hostList*. There are two values: either `read-write` or `read-only`.

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

- A host name
- An IP address
- A subnet mask

The set of all access configurations defines the access policy of the SNMP agent. A manager whose host is specified in a *hostList* and that identifies itself in one of the communities of the same configuration will be granted the permissions defined by the

corresponding *accessRights*. A manager's host can appear in several access configurations provided it is associated with a different community list. This will define different access communities with different rights from the same manager.

A manager whose host-community identification pair does not appear in any of the access configurations will be denied all access. This means that protocol data units (PDU) from this manager will be dropped without being processed.

Format of the Trap Group

The trap group specifies the hosts to which the agent will send traps if the `InetAddressAcl` mechanism is used. This group contains one or more trap community definitions.

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

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

```
{
    trap-community = trapCommunityName
    hosts = trapHostList
}
```

The *trapCommunityName* item specifies a single SNMP community string. It will be included in the traps sent to the hosts specified in the `hosts` item. SNMPv3 does not use the community string, so use IP addresses or the context name instead.

The *trapHostList* item specifies a comma-separated list of hosts. Each host must be identified by its name or complete IP address.

When the SNMP protocol adaptor is instructed to send a trap using the `InetAddressAcl` mechanism, it will send a trap to every host listed in the trap community definitions. If a host is present in more than one list, it will receive more than one trap, each one identified by its corresponding trap community.

SNMPv3 User-Based Access Control

The user-based access control implemented by SNMPv3 is based on contexts and user names. The users, contexts and associated security information controlling access to the agents in an SNMP session are defined in the `nma.uacl` file.

EXAMPLE 4–2 Example `nma.uacl.template` File

```
acl = {
# {
# context-names = TEST-CONTEXT
# access = read-write
# security-level = authNoPriv
# users = defaultUser
# }
}
```

In the `nma.uacl` file, you define the following:

- A list of context names, separated by commas. You can define a null context by declaring `context-names = null`
- The access level, which can be either `read-write` or `read-only`
- The security level, as follows:

`noAuthNoPriv`	No security mechanisms activated
`authNoPriv`	Authentication activated, with no privacy
`authPriv`	Both authentication and privacy activated

- A list of authorized users, separated by commas; an asterisk (`*`) opens access to all users.

By uncommenting the `acl` block in Example 4–2, you would limit access to MIBs in the `TEST-CONTEXT` context only, and grant read-write access to the user `defaultUser`. The security level in the file must also match that of user `defaultUser`. Therefore, any non-authenticated requests, any request with different security levels, or any requests from a user other than `defaultUser`, would be rejected.

Configuring SNMPv3 Security

Under SNMPv1 and SNMPv2, agents act as information servers, and IP-based access control is used to protect this information from unauthorized access. The SNMPv3 protocol provides much more sophisticated security mechanisms, implementing a user-based security model (USM). This model allows both authentication and encryption of the requests sent between agents and their managers, as well as user-based access control.

Note – The default NMA configuration is an example of an SNMPv3 configuration. Modify the security parameters to fit your security requirements.

You can add and remove users in the `nma.security` file as specified in "Managing Users in Security Files" on page 35.

Engine ID

Secure SNMPv3 communication requires that the SNMP engine ID, which is generated by the NMA for each node, is used to communicate with the NMA. The SNMP engine ID is unique for the SNMP domain. It is a hexadecimal string calculated from a concatenation of the following properties of the NMA on each node:

- Node CGTP address
- Communication port number
- IANA number. By default this is `42`.

The engine ID is stored in the `nma.security` file of each NMA. The engine ID may be substituted for another engine ID.

Context Name

The NMA MIB is not registered under the scope of any context.

Managing Users in Security Files

Every user that has access to an agent is represented by a `userEntry` line in each of the agent's security files.

You configure the `userEntry` as follows, with the parameters separated commas:

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

The only mandatory parameters are the engine ID and the user name. All the other parameters are optional.

The possible values for the parameters are as follows:

Engine ID	A local or remote SNMP engine, defined in one of the following ways: ■ The string `localEngineID`, to denote the local engine ■ A hexadecimal string, for example, `0x8000002a05819dcb6e00001f95` ■ A human-readable string used to generate an engine ID, providing any or all of the host name, port, and IANA number.
User name	Any human-readable string
Security name	Any human-readable string
Authentication algorithm	The following algorithms are permitted: ■ `usmHMACMD5AuthProtocol` ■ `usmHMACSHAAuthProtocol` ■ `usmNoAuthProtocol`
Authentication key	Any text password or any hexadecimal key starting with `0x`; for example, `0x0098768905AB67EFAA855A453B665B12`, of size: ■ 0 to 32 inclusive for HMACMD5 ■ 0 to 40 inclusive for HMACSHA
Privacy algorithm	The following algorithms are permitted: ■ `usmDESPrivProtocol` ■ `usmNoPrivProtocol` If no algorithm is specified, the default is `usmNoPrivProtocol`.

	Any text password or any hexadecimal key starting with `0x`; for example, `0x0098768905AB67EFAA855A453B665B12`, of size 0 to 32 inclusive If a hexadecimal string is provided, it must be a localized key
Storage type	A value of 3 denotes *non-volatile*, meaning that the user entry is flushed in the security file; any value other than 3 will be rejected, throwing an `IllegalArgumentException`
template	Can be either `true` or `false`: If `true`, the row is a template, not seen from USM MIB. This kind of user is used when cloning users. The default is `false`.

Users can also be managed through USM MIB access.

SNMP Manager Configuration Examples

This section contains three examples of SNMP configurations. The NMA implements the Notification MIB module specified by the Internet Engineering Task Force in RFC 2573, which is accessible from `http://www.ietf.org/rfc/rfc2573.txt`.

By default the NMA authorizes `localhost` to access its MIB using SNMPv1 or SNMPv2 on port `8085`. SNMP traps are sent using the mechanism described in the RFC 2573. Traps are sent by default to `localhost` on port `8086` using SNMPv2 parameters, as defined in the default RFC 2573 configuration files:

- `nma.params.txt`
- `nma.notifs.txt`
- `nma.targets.txt`

The RFC 2573 configuration files can be manually edited. Alternately, use the `com.sun.jdmk.snmp.rfc2573.manager.SnmpV3AppliMibRegistration` class, found in the `rfc2573mgr.jar` file. Use this class to dynamically register or unregister SNMP managers at runtime. Example 4–3 is a code snippet that uses this class to register a trap target on trap port `trapPort` of the host `localHost`. Traps are received using SNMPv3 parameters.

EXAMPLE 4–3 Using the `SnmpV3AppliMibRegistration` API

```
//Register the manager/params to the NMA
try {

System.out.println("Register the Manager to receive Traps using SNMPv3 " +
                   parameters");

// Register the SNMP Parameters V3
SnmpV3AppliMibRegistration.registerParams(session,
                                          "manager_paramsv3",
                                          3,
                                          3,
                                          "defaultUser",
                                          2);

// Register the Manager to receive traps with SNMPv3 parameters
SnmpV3AppliMibRegistration.registerTarget(session,
                                          "manager_targetv3",
                                          "1.3.6.1.6.1.1",
                                          localHost + "/"+ trapPort,
                                          10000,
                                          2,
                                          "trap",
                                          "manager_paramsv3");

}
catch(SnmpStatusException e) {
  System.out.println("ERROR in registration " + e.getMessage());
}
```

SNMPv2 Configuration Example

In this configuration the NMA MIB is accessed using SNMPv2 on port number `8085`. The SNMP manager is authorized to access the MIB located on host `10.8.1.253`. Traps are sent to the manager on port `8086` using SNMPv2, using the Notification MIB described in RFC 2573.

Example 4–4, Example 4–5, Example 4–6, Example 4–7, and Example 4–8 list the entries in the NMA configuration files that support this SNMP configuration.

EXAMPLE 4–4 Example Entries in `nma.properties` for SNMPv2

```
com.sun.nhas.ma.adaptors.snmp.enabled=true
com.sun.nhas.ma.adaptors.snmp.port=8085
com.sun.nhas.ma.adaptors.snmp.rfc2573.enabled=true
com.sun.nhas.ma.adaptors.snmp.rfc2573.v1v2set.enabled=true
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.addr.file=\
/etc/opt/SUNWcgha/nma.targets.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.params.file=\
```

EXAMPLE 4–4 Example Entries in `nma.properties` for SNMPv2 *(Continued)*

```
/etc/opt/SUNWcgha/nma.params.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.notification.file=\
/etc/opt/SUNWcgha/nma.notifs.txt
jdmk.acl.file=/etc/opt/SUNWcgha/nma.acl
```

EXAMPLE 4–5 Example Entries in `nma.acl` for SNMPv2

```
acl = {
 {
 communities = public, private
 access = read-only
 managers = 10.8.1.253
 }
 {
 communities = public, private
 access = read-write
 managers = 10.8.1.253
 }
}
```

EXAMPLE 4–6 Example Entry in `nma.targets.txt` for SNMPv2

```
targetsEntry=managerV2,snmpUDPDomain,10.8.1.253/8086,10000,2,trap,snmpV2,3
```

EXAMPLE 4–7 Example Entry in `nma.params.txt` for SNMPv2

```
paramsEntry=snmpV2,1,2,public,1,3
```

EXAMPLE 4–8 Example Entry in `nma.notifs.txt` for SNMPv2

```
notificationEntry=notif1,trap,1,3
```

SNMPv2 and SNMPv3 Hybrid Configuration Example

In this configuration the NMA is located at the CGTP address `10.8.3.18`. The NMA MIB can be accessed through SNMPv2 and SNMPv3 using port number `8085`. The manager that authorizes access to the MIB in SNMPv2 is located on host 10.8.1.253. The user `defaultUser` is authorized to access the MIB through SNMPv3 using the security parameters described in the `nma.security` file. Traps are sent to the manager on port `8086` using SNMPv2 and on port `8095` using SNMPv3. The notification MIB described in the RFC 2573 is used.

Example 4–9, Example 4–10, Example 4–11, Example 4–12, Example 4–13, Example 4–14, Example 4–15, and Example 4–16 list the entries in the NMA configuration files that support this SNMP configuration.

EXAMPLE 4–9 Example Entries in `nma.properties` for Hybrid Configuration

```
com.sun.nhas.ma.adaptors.snmp.enabled=true
com.sun.nhas.ma.adaptors.snmp.port=8085
com.sun.nhas.ma.adaptors.snmp.rfc2573.enabled=true
com.sun.nhas.ma.adaptors.snmp.rfc2573.v1v2set.enabled=true
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.addr.file=\
/etc/opt/SUNWcgha/nma.targets.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.params.file=\
/etc/opt/SUNWcgha/nma.params.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.notification.file=\
/etc/opt/SUNWcgha/nma.notifs.txt
jdmk.acl.file=/etc/opt/SUNWcgha/nma.acl
jdmk.uacl.file=/etc/opt/SUNWcgha/nma.uacl
```

EXAMPLE 4–10 Example Entries in `nma.security` for Hybrid Configuration

```
userEntry=localEngineID,defaultUser,null,usmHMACMD5AuthProtocol,mypasswd
localEngineBoots=23
localEngineID=0x8000002a050a08031200001f95
```

EXAMPLE 4–11 Example Entries in `nma.acl` for Hybrid Configuration

```
acl = {
 {
 communities = public, private
 access = read-only
 managers = 10.8.1.253
 }
 {
 communities = public, private
 access = read-write
 managers = 10.8.1.253
 }
}
```

EXAMPLE 4–12 Example Entries in `nma.uacl` for Hybrid Configuration

```
acl = {
 {
 context-names = null
 access = read-write
 security-level=authNoPriv
 users = defaultUser
 }
}
```

EXAMPLE 4–13 Example Entries in `nma.targets.txt` for Hybrid Configuration

```
targetsEntry=managerV2,snmpUDPDomain,10.8.1.253/8086,10000,2,trap,snmpV2,3
targetsEntry=managerV3,snmpUDPDomain,10.8.1.253/8095,10000,2,trap,snmpV3,3
```

EXAMPLE 4–14 Example Entries in `nma.params.txt` for Hybrid Configuration

```
paramsEntry=snmpV2,1,2,public,1,3
paramsEntry=snmpV3,3,3,defaultUser,2,3
```

EXAMPLE 4–15 Example Entry in `nma.notifs.txt` for Hybrid Configuration

```
notificationEntry=notif1,trap,1,3
```

EXAMPLE 4–16 Example Entry in `nma.security` for Hybrid Configuration

```
userEntry=10.8.3.18:8085,defaultUser,defaultUser,usmHMACMD5AuthProtocol,\
mypasswd
userEntry=localEngineID,defaultUser,defaultUser,usmHMACMD5AuthProtocol,\
mypasswd
localEngineBoots=26
localEngineID=0x8000002a05000000ef6540c3f9
```

SNMPv3 Configuration Example

In this configuration, the NMA MIB is accessed using SNMPv3 on port number `8085`. The manager authorized to access the MIB is located on host `10.8.1.253`. Traps are sent to the manager on trap port `8086`. In this case, the notification MIB is not used. Traps are always sent to trap port `8086` as defined in the `nma.properties` file and use only SNMPv2. The `nma.targets.txt`, `nma.params.txt`, and `nma.notifs.txt` files are not used in this configuration.

Example 4–17, Example 4–18, Example 4–19, and Example 4–20 list the entries in the NMA configuration files that support this SNMP configuration.

EXAMPLE 4–17 Example Entries in `nma.properties` for SNMPv3 Configuration

```
com.sun.nhas.ma.adaptors.snmp.enabled=true
com.sun.nhas.ma.adaptors.snmp.port=8085
com.sun.nhas.ma.adaptors.snmp.trap.port=8086
com.sun.nhas.ma.adaptors.snmp.rfc2573.enabled=false
com.sun.nhas.ma.adaptors.snmp.rfc2573.v1v2set.enabled=false
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.addr.file=\
/etc/opt/SUNWcgha/nma.targets.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.target.params.file=\
/etc/opt/SUNWcgha/nma.params.txt
com.sun.nhas.ma.adaptors.snmp.rfc2573.notification.file=\
/etc/opt/SUNWcgha/nma.notifs.txt
jdmk.acl.file=/etc/opt/SUNWcgha/nma.acl
```

EXAMPLE 4–17 Example Entries in `nma.properties` for SNMPv3 Configuration *(Continued)*

```
jdmk.uacl.file=/etc/opt/SUNWcgha/nma.uacl
```

EXAMPLE 4–18 Example Entries in `nma.security` for SNMPv3 Configuration

```
userEntry=localEngineID,defaultUser,null,usmHMACMD5AuthProtocol,mypasswd
localEngineBoots=23
localEngineID=0x8000002a050a08031200001f95
```

EXAMPLE 4–19 Example Entries in `nma.acl` for SNMPv3 Configuration

```
acl = {
 {
 communities = public, private
 access = read-only
 managers = 10.8.1.253
 }
 {
 communities = public, private
 access = read-write
 managers = 10.8.1.253
 }
}

trap = {
  {
  trap-community = public
  hosts = 10.8.1.253
  }
  {
  trap-community = private
  hosts = 10.8.1.253
  }
}
```

EXAMPLE 4–20 Example Entries in `nma.uacl` for SNMPv3 Configuration

```
acl = {
 {
 context-names = null
 access = read-write
 security-level=authNoPriv
 users = defaultUser
 }
}
```

CHAPTER 5

Manipulating the Cluster Using the NMA

The Node Management Agent (NMA) exposes methods for causing a switchover and manipulating the retry count for processes and process groups monitored by the Daemon Monitor.

This chapter contains the following sections:

Using the NMA to Initiate a Switchover

Methods of the `CmmMasterNodeMBean` can be used to check whether a switchover is possible, initiate the switchover, and then gauge the success of the switchover. The switchover is performed by the Cluster Membership Manager (CMM).

Because it is possible to set a timeout value for CMM operations, it is also possible that CMM operations might not be completed during the time allowed. If the timeout value is too short, some or all CMM operations will fail. For more information about this CMM behavior, see `cmm_connect`(3CMM).

Note – To disable the ability to perform remote operations on a cluster, set the `com.sun.nhas.ma.operation.flag` property in `nma.properties` to `false`.

Checking Whether the Foundation Services Are Ready for Switchover

To check whether a switchover is possible, invoke the `isSwitchOverReady` method. The `isSwitchOverReady` method takes no parameters, and returns a `boolean`.

Note – Even if the `isSwitchOverReady` method returns `true`, this does not guarantee that switchover will succeed. Switchover may not succeed, for example, if cluster readiness changes between the time when the `isSwitchOverReady` is invoked and the `switchOver` method is invoked.

Initiating a Switchover

To initiate a switchover, invoke the `switchOver` method. The `switchOver` method takes no parameters, and returns a `void`. Note that if this method is invoked using the floating external address the connection to the NMA might be broken prematurely and the `switchOver` method will never finish executing. Write code to detect and handle this eventuality.

Example of Switchover Using an HTTP Connector Client

The `NmaSwitchover` class, the code of which is listed below, can be used to provoke a switchover of the current master node. The mechanism used below (the `invoke()` method of the `HTTPConnectorClient` class) can be used to invoke the methods of the NMA MBeans.

EXAMPLE 5–1 `NmaSwitchover.java`

```
// java import
//
import java.net.InetAddress;

// jmx import
//
import javax.management.ObjectName;
import javax.management.MBeanException;

// jdmk import
//
import com.sun.jdmk.TraceManager;
import com.sun.jdmk.comm.HttpConnectorClient;
import com.sun.jdmk.comm.HttpConnectorAddress;
import com.sun.jdmk.comm.CommunicationException;
```

EXAMPLE 5–1 NmaSwitchover.java *(Continued)*

```
import com.sun.jdmk.comm.UnauthorizedSecurityException;

/**
 * This java client uses an HTTP connector client to establish a connection
 * to the NMA and perform a mastership switchover.
 *
 * To compile the client:
 *
 * javac NmaSwitchover.java
 *
 * Note: First ensure that the jar files specified in the chapter
 * 'Configuration Files, Dependencies and Requirements' of the
 * "Netra High Availability Suite Foundation Services 2.1 6/03
 * NMA Programming Guide" are in your CLASSPATH.
 *
 * To run the client:
 *
 * java NmaSwitchover <domain_name> <master_IP_address>
 * <HttpConnectorServer_port>
 *
 * For example:  java NmaSwitchover cluster_8 10.8.1.18 8081
 *
 * Note: This example must be run on a machine with access to the
 * cluster, for example, the cluster install server.
 *
 */
public class NmaSwitchover {

  public static void main(String argv[]) {

    try {

        /**
        * Debug
        * To activate the debug or trace mechanism from the command
        * line, use the syntax: java -DLEVEL_DEBUG NmaSwitchover
        * <arguments> or  java -DLEVEL_TRACE NmaSwitchover <arguments>
        *
        * For example:
        * java -DLEVEL_DEBUG NmaSwitchover cluster_6 10.6.1.1 8081
        *
        */

        TraceManager.parseTraceProperties();

        // Set the domain name of the cluster
        //
        String domain = "DefaultDomain";
        if (argv.length >=1) domain = argv[0];

        // Set the host name of the remote MBean server.
        //
        String agentHost = InetAddress.getLocalHost().getHostName();
```

EXAMPLE 5–1 NmaSwitchover.java *(Continued)*

```
 if (argv.length >= 2) agentHost = argv[1];

 // Set the port number of the remote connector server.
 //
 int agentPort = 8081;
 if (argv.length >= 3)
     agentPort = Integer.decode(argv[2]).intValue();

 System.out.println(">>> Connecting to " + agentHost +
            " using port number " + agentPort);

 // Set up the HTTP Connector Client.
 //
 HttpConnectorClient connector = new HttpConnectorClient();
 try {
   // Initialize communication with the remote MBean server.
   //
HttpConnectorAddress hca =
         new HttpConnectorAddress(agentHost,agentPort);
connector.connect(hca);
} catch (IllegalArgumentException e)
  {
  System.out.println("Connection exception! " +
        e.getMessage());
  } catch (CommunicationException e)
  {
    System.out.println("Connection exception! " +
        e.getMessage());
  } catch (UnauthorizedSecurityException e)
  {
    System.out.println("Connection exception! " +
        e.getMessage());
  }

  String[] iargs = {};
  String[] isig = {};

  String instanceName = domain +
     ".master:nhas-object=cluster_node";
  ObjectName node = new ObjectName(instanceName);
// Invoke the isSwitchOverReady() method to check that the
// cluster is in a condition to support switchover successfully.
// If this method returns true, invoke the switchover method.
// Note that this does not guarantee that the switchover operation
// will be successfully invoked, because the readiness of the
// cluster may change before the switchover can be performed.
//
  try {
    if (connector.invoke(node, "isSwitchOverReady",
            iargs, isig).equals(new Boolean(true)))
    {
   System.out.println("Performing switchover");
      // Switchover
```

EXAMPLE 5–1 NmaSwitchover.java *(Continued)*

```
            //
            connector.invoke(node, "switchOver", iargs, isig);
        } else
          {
            System.out.println("Cluster not ready for switchover");
          }
        } catch (MBeanException e)
        {
          System.out.println("Got an exception invoking switchover! " +
          e.getTargetException().getMessage());
        }

        // Terminate communication with the remote MBean server.
        //
        connector.disconnect();

        // Exit program
        //
        System.exit(0);

    } catch (Exception e) {
        System.out.println("Got an exception !" + e.getMessage());
        e.printStackTrace();
        System.exit(1);
        }
  }
}
```

Getting the CMM Status of All Cluster Nodes

To get the CMM status of all cluster nodes, invoke the `getAllNodeInfo` method. The `getAllNodeInfo` method takes no parameters, and returns a `CmmMemberInfo[]`.

Manipulating Daemon Monitor Retry Settings

The Daemon Monitor controls groups of processes and attempts to restart these processes if they fail. The number of times that the Daemon Monitor attempts to restart a group of failed processes can be set using the `updateMaxRetryCount` method of the `PmdNameTagStatisticsMBean`. When this method is invoked, the current count of the number of times that the Daemon Monitor has attempted to restart the process group is reset. When a group of processes has been successfully restarted, invoke the `resetRetryCount` method to ensure that the stipulated number of retries are attempted if the process group fails again.

CHAPTER 6

Carrier Grade Transport Protocol Statistics

For information about the Carrier Grade Transfer Protocol (CGTP) statistics that can be accessed from the NMA, see the following sections:

Introducing CGTP Statistics

The CGTP statistics collected by the NMA can be used to ascertain the degree of success with which CGTP is operating in a running cluster. Network traffic conditions might slow down or prohibit the arrival of duplicate packets, or a bottleneck might occur at the local or remote end of the CGTP link. CGTP statistics provide a measure of the success with which CGTP is operating. These statistics are also useful for diagnosing the source of bad performance or failure. Using the CGTP statistics collected by the NMA it is possible to measure the success of packet duplication and filtering, redundant link by redundant link, and pinpoint the source of slow performance or packet loss.

For more information about CGTP, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

Set the `com.sun.nhas.ma.cgtp.polling` property to specify the interval in milliseconds between sequential updates of CGTP information in the NMA.

CGTP Master Statistics

CgtpMasterMBean

The `CgtpMasterMBean` MBean exposes the CGTP master node view. This MBean provides lists of local and remote CGTP addresses. One MBean implementing the `CgtpMasterMBean` interface is instantiated on the cluster master node.

Getting All Local CGTP Addresses for Which Statistics are Available

To return the list of all local CGTP addresses for which statistics are available, invoke the `getLocalCgtpAddresses` method. The `getLocalCgtpAddresses` method takes no parameters, and returns a `String[]`. If statistics are not available, the method returns `null`.

Getting All Remote CGTP Addresses for Which Statistics are Available

To return the list of remote CGTP addresses for which statistics are available, invoke the `getRemoteCgtpAddresses` method. The `getRemoteCgtpAddresses` method takes no parameters, and returns a `String[]`. If statistics are not available, the method returns `null`.

CGTP Node Statistics

CgtpMBean

One instance of the `CgtpMBean` is instantiated on each node.

Getting All Local CGTP Addresses for Which Statistics Are Available

To return the list of all local CGTP addresses for which statistics are available, invoke the `getLocalCgtpAddresses` method. The `getLocalCgtpAddresses` method takes no parameters, and returns a `String[]`. If statistics are not available, the method returns `null`.

Getting All Remote CGTP Addresses for Which Statistics Are Available

To return the list of remote CGTP addresses for which statistics are available, invoke the `getRemoteCgtpAddresses` method. The `getRemoteCgtpAddresses` method takes no parameters, and returns a `String[]`. If statistics are not available, the method returns `null`.

`CgtpEmitterStatisticsMBean`

The `CgtpEmitterStatisticsMBean` MBean provides statistics about the packets a node has sent through the local subinterface, in the reliable link operated by CGTP from a local CGTP address to a remote CGTP address.

Getting the Number of Packets Sent Through Each Subinterface

To get the number of packets sent through each local subinterface taking part in the reliable link, invoke the `getSubInterfaceSentCount` method. The subinterfaces are placed in the same order as that used in `CgtpReliableLinkStatisticsMBean.getSubInterfaceAddresses()`. The `getSubInterfaceSentCount` method takes no parameters, and returns an `int[]`.

`CgtpFilterMBean`

This MBean interface exposes the statistics available on the CGTP filter. There is one MBean per node which provides information on the CGTP filter.

Getting the Number of Packets Not Received in Duplicate

To get the number of packets that have not been duplicated, invoke the `getFilterFailure` method. The `getFilterFailure` method takes no parameters, and returns a `long`.

Getting the Amount of Memory Currently Used by the Filter Module

To get the amount of memory used by the filter module, invoke the `getFilterMemory` method. The `getFilterMemory` method takes no parameters, and returns a `long`.

Getting the Number of Packets Successfully Received

To get the number of packets successfully received and successfully filtered, invoke the `getFilterSuccess` method. The `getFilterSuccess` method takes no parameters, and returns a `long`.

Getting the Number of Filter Tables Used by the Filter Module

To get the current number of filter tables used by the filter module, invoke the `getFilterTables` method. The `getFilterTables` method takes no parameters, and returns an `int`.

Getting the Number of Hash Table Collisions

To get the number of collisions that have occurred in the hash table, invoke the `getHashCollisions` method. The `getHashCollisions` method takes no parameters, and returns an `int`.

Getting the Number of Direct Hash Table Entries

To get the number of direct entries in the hash table, invoke the `getHashDirect` method. The `getHashDirect` method takes no parameters, and returns an `int`.

Getting the Number of Hash Table Entries

To get the number of entries in the hash table, invoke the `getHashEntries` method. The `getHashEntries` method takes no parameters, and returns an `int`.

Getting the Number of Packets Not Received

To get the number of packets for which filtering has failed on the local node for each interface, invoke the `getInterfaceFilterFailure` method. The `getInterfaceFilterFailure` method takes no parameters, and returns a `long[]`.

Getting the Maximum Amount of Memory Used by the Filter Module

To get the highest amount of memory used by the filter, invoke the `getMaxFilterMemory` method. The `getMaxFilterMemory` method takes no parameters, and returns a `long`.

Getting the Maximum Number of Filter Tables Used by the Filter Module

To get the maximum number of filter tables used by the filter module, invoke the `getMaxFilterTables` method. The `getMaxFilterTables` method takes no parameters, and returns an `int`.

Getting the Maximum Number of Ticks Allowed Before Duplicate Arrives

To get the maximum number of ticks allowed until the duplicate of a packet is received and the packet is validated as successfully filtered, invoke the `getMaxTickPremiumToDuplicate` method. The `getMaxTickPremiumToDuplicate` method takes no parameters, and returns an `int`. If the duplicate packet does not arrive within the tick period, the package will not be validated as successfully filtered.

Getting the Number of Packets Waiting for Duplicate Reception

To get the number of packets for which no duplicate has yet been received, invoke the `getPremiumPacket` method. The `getPremiumPacket` method takes no parameters, and returns an `int`.

`CgtpReceiverStatisticsMBean`

The `CgtpReceiverStatisticsMBean` MBean provides statistics about the packets received by this node through the reliable link operated by CGTP from a remote CGTP address to a local CGTP address.

Getting the Number of Packets Not Successfully Filtered

To return the number of packets not successfully filtered on reception, invoke the `getFilterFailureCount` method. The `getFilterFailureCount` method takes no parameters, and returns an `int`.

Getting the Number of Packets Successfully Filtered

To return the number of packets successfully filtered on reception, invoke the `getFilterSuccessCount` method. The `getFilterSuccessCount` method takes no parameters, and returns an `int`.

Getting the Number of Packets Received Through Each Subinterface

To return the number of packets received through each local subinterface taking part in the reliable link, invoke the `getSubInterfaceReceivedCount` method. The subinterfaces are placed in the same order as that used in `CgtpReliableLinkStatisticsMBean.getSubInterfaceAddresses()`. The `getSubInterfaceReceivedCount` method takes no parameters, and returns an `int[]`.

`CgtpReliableLinkStatisticsMBean`

The `CgtpReliableLinkStatisticsMBean` MBean lists the addresses of the subinterfaces, and the reliable link addresses currently in use by the Reliable Transport Service.

Getting the Remote Subinterface Addresses

To get the remote interface addresses used by CGTP to send and receive packets, invoke the `getGatewayAddresses` method. The number of redundant links making up the reliable link is limited to two. The array elements are ordered identically to the subinterface. The `getGatewayAddresses` method takes no parameters, and returns a `String[]`.

Getting Local End Reliable Link CGTP Addresses

To get the local CGTP address at the local end of the reliable link for which these statistics are provided, invoke the `getLocalCgtpAddress` method. The `getLocalCgtpAddress` method takes no parameters, and returns a `String`.

Getting Remote End Reliable Link CGTP Addresses

To return the remote CGTP address at the remote end of the reliable link for which these statistics are provided, invoke the `getRemoteCgtpAddress` method. The `getRemoteCgtpAddress` method takes no parameters, and returns a `String`.

Getting Local Subinterface Addresses

To return the underlying local subinterface addresses used by CGTP to send and receive packets, invoke the `getSubInterfaceAddresses` method. The `getSubInterfaceAddresses` method takes no parameters, and returns a `String[]`. The number of redundant links making up the reliable link is limited to two. The array elements are ordered identically to those of the gateway.

CHAPTER 7

Daemon Monitor Statistics

This chapter describes the Daemon Monitor statistics that can be accessed from the NMA.

This chapter contains the following sections:

Example of Accessing Statistics Using an HTTP Client

The `NmaMasterNametags` example, the code of which is listed in Example 7–1, queries the `PmdMasterStatisticsMBean` for the list of daemon monitor nametags active on the master node. The mechanism used below (the `invoke()` method of the `HTTPConnectorClient` class) can be used to invoke the methods of the NMA MBeans and query the NMA for statistics and information.

EXAMPLE 7–1 `NmaMasterNametags.java`

```
/*
* @(#)file      NmaMasterNametags.java
* @(#)author    Sun Microsystems, Inc.
* @(#)version   1.2
* @(#)date      02/06/06
*
* Copyright 2002 Sun Microsystems, Inc. All rights reserved.
* This software is the proprietary information of Sun Microsystems, Inc.
* Use is subject to license terms.
```

EXAMPLE 7–1 NmaMasterNametags.java *(Continued)*

```
*
* Copyright 2002 Sun Microsystems, Inc. Tous droits réservés.
* Ce logiciel est proprieté de Sun Microsystems, Inc.
* Distribué par des licences qui en restreignent l'utilisation.
*/

// java import
//
import java.net.InetAddress;

// jmx import
//
import javax.management.ObjectName;
import javax.management.MBeanException;

// jdmk import
//
import com.sun.jdmk.TraceManager;
import com.sun.jdmk.comm.HttpConnectorClient;
import com.sun.jdmk.comm.HttpConnectorAddress;
import com.sun.jdmk.comm.CommunicationException;
import com.sun.jdmk.comm.UnauthorizedSecurityException;


/**
 * This java client uses an HTTP connector client to establish a connection
 * to the Master NMA and retrieve all Nametags.
 *
 * To compile the client:
 *
 * javac NmaMasterNametags.java
 *
 * Note: First ensure that the jar files specified in the chapter
 * 'Configuration Files, Dependencies and Requirements' of the
 * "Netra High Availability Suite Foundation Services 2.1 6/03
 * NMA Programming Guide" are in your CLASSPATH.
 *
 * To run the client:
 *
 * java NmaMasterNametags <domain_name> <master_IP_address>
 *  <HttpConnectorServer_port>
 *
 * For example:  java NmaMasterNametags cluster_8 10.8.1.18 8081
 *
 * Notes:
 * 1) This example must be run on a machine with access to the
 * cluster, for example, the cluster install server.
 * 2) The second parameter can also be the master floating address, for
 * example, 10.8.1.1
 *
 */

public class NmaMasterNametags {
```

EXAMPLE 7–1 NmaMasterNametags.java *(Continued)*

```
    public static void main(String argv[]) {

        try {

            /**
             * Debug
             * To activate the debug or trace mechanism from the command
             * line, use the syntax:
             * java -DLEVEL_DEBUG NmaMasterNametags <arguments> or
             * java -DLEVEL_TRACE NmaMasterNametags <arguments>
             *
             * For example:
             * java -DLEVEL_DEBUG NmaMasterNametags cluster_6 10.6.1.1 8081
             *
             */

            TraceManager.parseTraceProperties();

            // Set the domain name of the cluster
            //
            String domain = "DefaultDomain";
            if (argv.length >=1) domain = argv[0];

            // Set the host name of the remote MBean server.
            //
            String agentHost = InetAddress.getLocalHost().getHostName();
            if (argv.length >= 2) agentHost = argv[1];

            // Set the port number of the remote connector server.
            //
            int agentPort = 8081;
            if (argv.length >= 3)
                agentPort = Integer.decode(argv[2]).intValue();

            System.out.println(">>> Connecting to " + agentHost +
            " using port number " + agentPort);

            // Set up the HTTP Connector Client.
            //
            HttpConnectorClient connector = new HttpConnectorClient();

            try {
                // Initialize communication with the remote MBean server.
                //
                HttpConnectorAddress hca =
                new HttpConnectorAddress(agentHost,agentPort);
                connector.connect(hca);
            } catch (IllegalArgumentException e) {
                System.out.println("Connection exception! " +
                e.getMessage());
            } catch (CommunicationException e) {
                System.out.println("Connection exception! " +
```

EXAMPLE 7–1 NmaMasterNametags.java *(Continued)*

```
            e.getMessage());
        } catch (UnauthorizedSecurityException e) {
            System.out.println("Connection exception! " +
            e.getMessage());
        }

        // Get Nametags
        //

        String[] iargs = {};
        String[] isig = {};

        String instanceName = domain + ".master:nhas-object=pmd_stats";
        ObjectName node =
        new ObjectName(instanceName);
        try {
            // Attempt to invoke getNameTags()
            //
            String[] nt = (String[])
            connector.invoke(node, "getNameTags", iargs, isig);

            System.out.println("Node " + argv[0] +
            " is running process groups:");
            // Print each element of the array returned by getNameTags()
            // to the standard output.  Each element is a nametag
            // managed by the daemon monitor
            //
            for (int i = 0; i < nt.length; i++) {
                System.out.println(nt[i]);
            }
        } catch (MBeanException e) {
            System.out.println("Got an exception invoking " +
            "getNameTags()! " + e.getTargetException().getMessage());
        }

        // Terminate communication with the remote MBean server.
        //
        connector.disconnect();

        // Exit program
        //
        System.exit(0);

    } catch (Exception e) {
        System.out.println("Got an exception !" + e.getMessage());
        e.printStackTrace();
        System.exit(1);
    }
  }
}
```

Introducing Daemon Monitor Statistics

The Daemon Monitor statistics are useful in maintaining awareness of processes that fail, and processes that are unable to restart within the allowed number of retries. Access to the PIDs of the processes allows for the monitoring of these processes using standard Solaris operating system commands.

Note – Daemon Monitor statistics are cached. The `com.sun.nhas.ma.pmd.cache.validity` and `com.sun.nhas.ma.pmd.polling` properties in the `nma.properties` file control the Daemon Monitor polling interval and cache data validity period. If the values of these properties are set too low, the cache might be refreshed before all statistics cached in the previous polling period are read. The default values should be sufficient in most cases.

See "Daemon Monitor" in *Netra High Availability Suite Foundation Services 2.1 6/03 Overview* for more information about the Daemon Monitoring service.

Daemon Monitor Master Statistics

This section describes the Daemon Monitor statistics available from the NMA on the master node.

`PmdMasterStatisticsMBean`

The `PmdMasterStatisticsMBean` MBean provides the nametags of all daemons currently being monitored.

Getting All Nametags

To return all the nametags managed by the NMA, invoke the `getNameTags` method. The `getNameTags` method takes no parameters, and returns a `String[]`.

Daemon Monitor Node Statistics

This section describes the Daemon Monitor statistics collected by the NMA on each peer node.

PmdStatisticsMBean

The PmdStatisticsMBean provides a list of all the nametags monitored by the Daemon Monitor.

Getting All Nametags

To return all the nametags managed by the Daemon Monitor, invoke the getNameTags method. The getNameTags method takes no parameters, and returns a String[].

PmdNameTagStatisticsMBean

The PmdNameTagStatisticsMBean MBean provides information about the number of attempts that can be made to restart a daemon, and the number of attempts that have already been made. This MBean is the source of:

- A NhasPmdMaxRetriesNotification, which is sent whenever the maximum allowed number of retry attempts is exceeded.
- A AttributeValueChangeNotification, which is sent whenever the number of allowed retry attempts is changed.
- A NhasPmdNewNameTagNotification, which is sent whenever the Daemon Monitor creates a new nametag.
- A NhasPmdNewNameTagNotification, which is sent whenever the Daemon Monitor removes a nametag from the collection.

One instance of this MBean is instantiated for each Daemon Monitor by the Daemon Monitor service.

Getting the Daemon Monitor Nametag

To get the nametag that the PmdNameTagStatisticsMBean MBean is providing data on, invoke the getNameTag method. The getNameTag method takes no parameters, and returns a String.

Getting the PIDs Associated With a Nametag

To get the list of process IDs associated with this nametag, invoke the `getPidList` method. The `getPidList` method takes no parameters, and returns an `int[]`.

Getting the Daemon Monitor Maximum Retries

To get the maximum number of restart retries allowed for this nametag, invoke the `getMaxRetryCount` method. The `getMaxRetryCount` method takes no parameters, and returns an `int`.

Getting the Number of Retries for a Nametag

To number of restart retries already attempted for this nametag, invoke the `getRetryCount` method. The `getRetryCount` method takes no parameters, and returns an `int`.

CHAPTER 8

Reliable NFS Statistics

This chapter describes the Reliable NFS statistics that can be accessed from the NMA.

This chapter contains the following sections:

Introducing Reliable NFS Statistics

The Reliable NFS statistics collected by the NMA provide a view on the current state of replication in the cluster, node by node, reliable link by reliable link. Reliable NFS statistics are available only on the master node and the vice master node.

See "File Sharing and Data Replication" in the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview* for more information.

Reliable NFS Master Statistics

This section describes the Reliable NFS statistics collected by the NMA running on the master node.

RnfsMasterReplicatedSliceMBean

The RnfsMasterReplicatedSliceMBean MBean models a Reliable NFS replicated slice. Each slice is composed of a primary partition and a secondary partition. One instance of this MBean is instantiated for each replicated slice mounted on either the master or the vice master node.

Getting the Completed Recovery Percentage

To get the percentage of segments of the slice that has been resynchronized, invoke the getCompletedRecoveryPercentage method. The getCompletedRecoveryPercentage method takes no parameters, and returns a float. This information is meaningful if the primary slice of this MBean is mounted on the host running the agent. No statistics are provided for distant primary MBean.

Getting the Percentage of Segments Requiring Recovery

To get the percentage of segments of the slice that require recovery, invoke the getNeededRecoveryPercentage method. The getNeededRecoveryPercentage method takes no parameters, and returns a float which indicates the percentage of segments that require recovery. This information is meaningful if the primary slice of this MBean is mounted on the host running the agent. No statistics are provided for distant primary MBean.

Reliable NFS Node Statistics

This section describes the Reliable NFS statistics collected by the NMA running on each peer node.

RnfsStatisticsMBean

The RnfsStatisticsMBean MBean provides global Reliable NFS statistics. One instance of this MBean is instantiated on each master eligible node in the cluster. This MBean only provides statistics about Reliable NFS on the node on which it is running.

Getting the Primary Slice

To return the primary dual copy slice file name and slice name, invoke the getPrimarySlice method. The getPrimarySlice method takes no parameters, and returns a Slice.

Getting the Secondary Slice

To return the secondary dual copy slice file name and slice name, invoke the `getSecondarySlice` method. The `getSecondarySlice` method takes no parameters, and returns a `Slice`.

`RnfsMasterStatisticsMBean`

The `RnfsMasterStatisticsMBean` MBean provides Reliable NFS statistics on the master node.

Getting the Names of All Primary Files on the Local Host

To get an array of the names of the primary slices mounted on the local host, invoke the `getPrimaryFiles` method. The `getPrimaryFiles` method takes no parameters, and returns a `String[]`.

Getting the Names of All Secondary Files on the Local Host

To return an array of the names of the secondary slices mounted on the local host, invoke the `getSecondaryFiles` method. The `getSecondaryFiles` method takes no parameters, and returns a `String[]`.

`RnfsReplicatedSliceMBean`

The `RnfsReplicatedSliceMBean` MBean models a Reliable NFS slice. One instance of this MBean is instantiated for each replicated slice mounted on each master-eligible node.

Getting the Completed Recovery Percentage

To get the percentage of segments of the partition that has been resynchronized, invoke the `getCompletedRecoveryPercentage` method. The `getCompletedRecoveryPercentage` method takes no parameters, and returns a `float`. This information is meaningful if the primary slice of this MBean is mounted on the host running the agent. No statistics are provided for a distant primary MBean.

Getting the Dual Copy Status

To get the current status of the dual copy as a `DualCopyStatusEnum` value, invoke the `getDualCopyStatus` method. The `getDualCopyStatus` method takes no parameters, and returns a `DualCopyStatusEnum`. This information is meaningful if the primary slice of this MBean is mounted on the host running the agent. No statistics are provided for a distant primary MBean.

Getting the Link Status

To find out if replication is enabled, disabled, or in progress, invoke the `getLinkStatus` method. The `getLinkStatus` method takes no parameters, and returns a `LinkStatusEnum`.

The `LinkStatusEnum` can have one of the following values:

ENABLED	Replication is enabled.
RESYNC	A synchronization is in progress.

Getting the Percentage of Segments Requiring Recovery

To indicates the percentage of segments of the partition that require recovery, invoke the `getNeededRecoveryPercentage` method. The `getNeededRecoveryPercentage` method takes no parameters, and returns a `float`. This information is meaningful if the primary slice of this MBean is mounted on the host running the agent. No statistics are provided for a distant primary MBean.

Getting the Primary Slice

To returns the primary dual copy slice file name and slice name, invoke the `getPrimarySlice` method. The `getPrimarySlice` method takes no parameters, and returns a `Slice`.

Getting the Secondary Slice

To return the secondary dual copy slice file name and slice name, invoke the `getSecondarySlice` method. The `getSecondarySlice` method takes no parameters, and returns a `Slice`.

Getting the Names of All Primary Files on the Local Host

To get an array of the names of the primary slices mounted on the local host, invoke the `getPrimaryFiles` method. The `getPrimaryFiles` method takes no parameters, and returns a `String[]`.

Getting the Names of All Secondary Files on the Local Host

To return an array of the names of the secondary slices mounted on the local host, invoke the `getSecondaryFiles` method. The `getSecondaryFiles` method takes no parameters, and returns a `String[]`.

CHAPTER 9

Cluster Membership Manager Statistics

This chapter describes the Cluster Membership Manager (CMM) statistics that can be accessed from the NMA.

This chapter contains the following sections:

Introducing CMM Statistics

The CMM statistics collected by the NMA provide the role and status of each node in the cluster.

When a direct link is configured between the master-eligible nodes the NMA can monitor the following statistics:

- The number of times that the vice-master node has requested to become the master node.
- The state of the direct link. The state can be *up* or *down*.

For information about the direct link, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

Because it is possible to set a timeout value for CMM operations, it is also possible that CMM operations may not be completed during the time allowed. If the timeout value is too short, some or all CMM operations will fail. For more information about this CMM behavior, see `cmm_connect`(3CMM).

CMM Master Statistics

CmmMasterStatisticsMBean

The CmmMasterStatisticsMBean MBean interface makes master state information available. One MBean implementing the CmmMasterStatisticsMBean interface is instantiated on the CMM cluster master node.

Getting the Average Time Between Node Starts

To get the average time in seconds between nodes when starting the CMM, invoke the getAverageElectionDelay method. This information can be used for tuning the master election mechanism. The getAverageElectionDelay method takes no parameters, and returns an int.

Getting the Number of CMM Clients

To get the number of CMM clients currently connected, invoke the getClientCount method. The getClientCount method takes no parameters, and returns an int. This information is available on all nodes.

Getting the CMM Lifetime

To get the lifetime of the CMM on this node, expressed in the number of seconds since boot, invoke the getCmmUpTime method. The getCmmUpTime method takes no parameters, and returns an int. This information is available on all nodes.

Getting the Number of Node Elections

To get the number of elections processed on the platform, invoke the getElectionCount method. The getElectionCount method takes no parameters, and returns an int.

Getting the Longest Interval Between Node Starts

To get the maximum time in seconds between nodes when starting the CMM, invoke the getMaxElectionDelay method. The getMaxElectionDelay method takes no parameters, and returns an int. This information can be used for tuning the election mechanism.

Getting the Shortest Interval Between Node Starts

To get the minimum time in seconds between nodes when starting the CMM, invoke the `getMinElectionDelay` method. The `getMinElectionDelay` method takes no parameters, and returns an `int`. This information can be used for tuning the election mechanism.

Getting the Number of Nodes in the Cluster

To get the number of nodes acknowledged by the master node as being present in the cluster, invoke the `getPresentNodeCount` method. The `getPresentNodeCount` method takes no parameters, and returns an `int`.

Getting the Number of Outstanding CMM Requests

To get the number of requests currently outstanding, invoke the `getRequestCount` method. The `getRequestCount` method takes no parameters, and returns an `int`. This information is available on all nodes.

Getting the Switchover Count

To get the number of switchovers performed, invoke the `getSwitchOverCount` method. The `getSwitchOverCount` method takes no parameters, and returns an `int`.

CMM Node Statistics

`ClusterNodeMBean`

The `ClusterNodeMBean` MBean interface exposes the CMM view of the local node on which the agent runs. This MBean interface makes the CMM state information of the current node available. One MBean implementing the `ClusterNodeMBean` interface is instantiated in each management agent in the cluster. The MBean that implements this interface is the emitter of the `java.com.sun.nhas.ma.cmm.NhasCmmNotification`.

Getting a Node's CGTP Address

To return the CGTP address of a node, invoke the `getCgtpAddress` method. The `getCgtpAddress` method takes no parameters, and returns a `String`. This is not a symbolic name for the node. Having the IP address in dot-notation makes it possible to avoid translation into IP format.

Getting the Domain ID of the Cluster that a Node is Eligible to Join

To return the domain ID of the cluster that a node is eligible to join, invoke the `getDomainId` method. The `getDomainId` method takes no parameters, and returns an `int`. The domain ID identifies the cluster that the node can join. A cluster is composed of nodes that have the same domain ID. Two nodes running incompatible versions of a software package must have different domain IDs. Nodes can only belong to one cluster. The ID of that cluster is the domain ID of the current node as defined in the CMM configuration. Because the CMM is only aware of what occurs in its own cluster, the domain ID field will be the same for all nodes reachable from this node. This information is useful when interpreting CMM debug traces, which will refer to this domain ID when necessary.

Getting the Time Since Node Was Last Rebooted

To return the incarnation number, invoke the `getIncarnationNumber` method. The incarnation number is computed locally by each node. The value of the incarnation number is the time of the last reboot expressed in the number of seconds since epoch (01/01/1970). The `getIncarnationNumber` method takes no parameters, and returns a `long`.

Getting the CMM Membership Role of a Node

To get the membership role of this node, invoke the `getMembershipRole` method. This information is extracted from the `CmmStateFlag`. The `getMembershipRole` method takes no parameters, and returns a `com.sun.nhas.ma.cmm.CmmMembershipRoleEnum`, which is one of the following values:

MASTER	The node is the current cluster master node
VICEMASTER	The node is the current vice-master node
IN_CLUSTER	The node is a regular node in the cluster
OUT_OF_CLUSTER	The node is down

Getting the Node ID

To return the unique ID that identifies this node within the cluster, invoke the `getNodeId` method. This information is useful when interpreting CMM debug traces, which refer to this ID. The `getNodeId` method takes no parameters, and returns an `int`.

Getting the Node Name

To get a human-readable String that uniquely identifies this node within the cluster, invoke the `getNodeName` method. The name is not intended to be a parameter to be passed to system calls. It is intended to be used to format display messages. For instance, this name could be formatted so as to refer to the position of the card on a shelf so as to make it easy for an operator to locate and replace it in case of failure. The `getNodeName` method takes no parameters, and returns a `String`.

Getting the Node Boot Image ID

To return the ID of the current boot image used by this node, invoke the `getSoftwareLoadId` method. Since CMM is only aware of the domain ID, if two nodes run two incompatible boot images they must have different domain IDs and different software load IDs. The `getSoftwareLoadId` method takes no parameters, and returns a `String`.

Getting the CMM State Flags of a Node

To return the membership state flags of a node, invoke the `getStateFlags` method. These flags are a concatenation of the administrative attributes, the membership roles and the qualification as seen from the CMM's perspective. The `getStateFlags` method takes no parameters, and returns a `CmmStateFlag`.

`CmmStatisticsMBean`

The `CmmStatisticsMBean` MBean provides statistics about the service performed by the CMM. Some of these statistics will only be available on the master node. Others will be available on each node in the cluster. One MBean instance is instantiated for each NMA on the cluster.

Getting the Average Time Taken To Start CMM Services

To get the average time in seconds between nodes when starting the CMM, invoke the `getAverageElectionDelay` method. The `getAverageElectionDelay` method takes no parameters, and returns an `int`. This information can be used for tuning the election mechanism. This information is available on the master node only.

Getting the Number of Master Elections Performed on a Node

To get the number of elections processed on the platform, invoke the `getElectionCount` method. The `getElectionCount` method takes no parameters, and returns an `int`. This information is available on the master node only.

Getting the Maximum Time Taken to Elect a Master Node

To get the maximum time in seconds between nodes when starting the CMM, invoke the `getMaxElectionDelay` method. The `getMaxElectionDelay` method takes no parameters, and returns an `int`. This information can be used for tuning the election mechanism. This information is available on the master node only.

Getting the Minimum Time Taken to Elect a Master Node

To get the minimum time in seconds between nodes when starting the CMM, invoke the `getMinElectionDelay` method. The `getMinElectionDelay` method takes no parameters, and returns an `int`. This information can be used for tuning the election mechanism. This information is available on the master node only.

Getting the Number of Nodes Present

To get the number of nodes acknowledged by the master node as being present in the cluster, invoke the `getPresentNodeCount` method. The `getPresentNodeCount` method takes no parameters, and returns an `int`. This information is available on the master node only.

Getting the Number of Switchovers Performed

To get the number of switchovers performed, invoke the `getSwitchOverCount` method. The `getSwitchOverCount` method takes no parameters, and returns an `int`. This information is available on the master node only.

Getting the Number of Currently Connected CMM Clients

To get the number of CMM clients currently connected, invoke the `getClientCount` method. The `getClientCount` method takes no parameters, and returns an `int`. This information is available on all nodes.

Getting the Number of Outstanding Requests

To get the number of requests currently outstanding, invoke the `getRequestCount` method. The `getRequestCount` method takes no parameters, and returns an `int`. This information is available on all nodes.

Getting the Lifetime of the CMM on a Node

To get the lifetime of the CMM on this node, expressed in the number of seconds since boot, invoke the `getCmmUpTime` method. The `getCmmUpTime` method takes no parameters, and returns an `int`. This information is available on all nodes.

CHAPTER 10

Receiving Notifications

This chapter explains the NMA notification mechanism and describes the NMA notifications in detail.

This chapter contains the following sections:

Registering to Receive Notifications

For information and instructions about writing and registering a notification listener, see the *Java Dynamic Management Kit 5.0 Tutorial*. Note that this information applies to the NMA only and is separate from the process of registering for notifications sent by the CMM. For information on these, see "Receiving and Handling Change Notifications" in the *Netra High Availability Suite Foundation Services 2.1 6/03 CMM Programming Guide*.

NhasCmmNotification

A `NhasCmmNotification` notification is sent by the `ClusterNodeMBean` when a node leaves or joins the cluster, or when a failover or switchover occurs. In addition to the standard notification information, this notification contains the following information:

`type` The possible types for this notification are listed below.

`source` The `ObjectName` of the node.

The notification type is one of the following:

MASTER	The node is now the cluster master node
VICEMASTER	The node is now the cluster vice-master node
IN_CLUSTER	The node is now part of the cluster
OUT_OF_CLUSTER	The node is no longer part of the cluster

NhasPmdMaxRetriesNotification

A NhasPmdMaxRetriesNotification notification is sent by the PmdStatisticsMBean when the maximum number of retries has been reached for a nametag. In addition to the standard notification information, this notification contains the following information:

source	The PmdStatisticsMBean ObjectName.
maxRetry	The maximum retry number that was exceeded.

The MAX_RETRIES field of this notification contains the name of the nametag that reached its maximum number of retries limit.

NhasPmdAttributeChangeNotification

A javax.management.AttributeChangeNotification is sent by the PmdNameTagStatisticsMBean when either of the following conditions is true:

- The number of allowed retries changes
- The retry counter is reset

In addition to the standard notification information, this notification contains the following information:

type	ATTRIBUTE_CHANGE
source	PmdNameTagStatisticsMBean
attributeName	Either RetryCount if the retry counter has been reset, or MaxRetryCount if the maximum number of retries allowed has changed.
oldValue	The old number of retries allowed
newValue	The new number of retries allowed

The nametag field contains the name of the nametag that has been affected.

NhasPmdNewNameTagNotification

A `NhasPmdNewNameTagNotification` is sent whenever the Daemon Monitor creates a new nametag. This notification contains the field `NEW_NAMETAG` which contains the name of the new nametag.

NhasPmdRemoveNameTagNotification

A `NhasPmdRemoveNameTagNotification` is sent whenever the Daemon Monitor removes a nametag from the collection. This notification contains the field `REMOVE_NAMETAG` which contains the name of the nametag that was removed.

Registering to Receive SNMP Traps

For a Java DMK SNMP manager to receive SNMP traps, an implementation of the `SnmpTrapListener` class must be registered on the SNMP trap port. Example 10–1 is an implementation of the `SnmpTrapListener` that listens for SNMPv1, SNMPv2 and SNMPv3 traps. Example 10–2 is a code snippet that registers the `TrapListenerImpl` class as a trap listener on trap port `trapPort`. The `TrapListenerImpl` class prints the details of all the traps it receives to the standard output.

EXAMPLE 10–1 Implementation of the `SnmpTrapListener` Class

```
class TrapListenerImpl implements SnmpTrapListener {

  public void processSnmpTrapV1(SnmpPduTrap trap) {
    System.out.println("NOTE: TrapListenerImpl received trap V1:");
    System.out.println("\tGeneric " + trap.genericTrap);
    System.out.println("\tSpecific " + trap.specificTrap);
    System.out.println("\tTimeStamp " + trap.timeStamp);
    System.out.println("\tAgent adress " + trap.agentAddr.stringValue());
  }

  public void processSnmpTrapV2(SnmpPduRequest trap) {
    System.out.println("NOTE: TrapListenerImpl received trap V2:");

    SnmpPdu pdu = trap.getResponsePdu();
    System.out.println("\tFrom Address" + pdu.address.getHostAddress());
  }

  public void processSnmpTrapV3(SnmpScopedPduRequest trap) {
    System.out.println("NOTE: TrapListenerImpl received trap V3:");
    System.out.println("\tContextEngineId : " +
    SnmpEngineId.createEngineId(trap.contextEngineId));
```

EXAMPLE 10–1 Implementation of the `SnmpTrapListener` Class *(Continued)*

```
    System.out.println("\tContextName : " + new String(trap.contextName));
    System.out.println("\tVarBind list :");
    for (int i = 0; i < trap.varBindList.length; i++) {
      System.out.println("oid : " + trap.varBindList[i].getOid() +
      " val : " + trap.varBindList[i].getSnmpValue());
    }
  }
```

The following code snippet registers the `TrapListenerImpl` class as a trap listener on trap port `trapPort`.

EXAMPLE 10–2 Registering a Trap Listener

```
System.out.println("Creating the trap listener on trapPort = " + trapPort);

// Create the Trap listener
TrapListenerImpl trapListener = new TrapListenerImpl();
SnmpEventReportDispatcher trapAgent = null;

try{
  trapAgent = new SnmpEventReportDispatcher(trapPort);
} catch (SocketException e) {
  System.out.println("ERROR Creating the trapListener " + e.getMessage());
}

// Start the Event Report dispatcher
new Thread(trapAgent).start();

// Add the trap listener on the Event report dispatcher
trapAgent.addTrapListener(trapListener);

System.out.println("Created the trap listener");
```

APPENDIX **A**

MBean Naming Conventions

This appendix describes the syntax of the MBean naming conventions.

All NMA MBeans are named according to conventions to allow easy location and manipulation. To identify an MBean, use an `ObjectName` in the format *domainID* : *MBeanName*. The *NodeID* is the unique ID given to each node in the cluster.

This appendix contains the following sections:

Nodes and Services

`ClusterNodeMBean`
`  nhas-object=cluster_node,node=`*NodeID*

`CmmMasterNodeMBean`
`  nhas-object=cluster_node`

`NhasSwitchOverService`
`  nhas-object=switchover,node=`*NodeID*

Cluster Membership Manager

`CmmMasterStatisticsMBean`
`nhas-object=cmm_stats`

`CmmStatisticsMBean`
`nhas-object=cmm_stats,node=`*NodeID*

Reliable NFS

`RnfsStatisticsMBean`
`nhas-object=rnfs_stats,node=`*NodeID*

`RnfsMasterStatisticsMBean`
`nhas-object=rnfs_stats`

`RnfsReplicatedSliceMBean`
`nhas-object=rnfs_stats,node=`*NodeID*`,file=`*SliceName*

Daemon Monitor

`PmdMasterStatisticsMBean`
`nhas-object=pmd_stats`

`PmdStatisticsMBean`
`nhas-object=pmd_stats,node=`*NodeID*

`PmdNameTagStatisticsMBean`
`nhas-object=pmd_stats,node=`*NodeID*`,nametag=`*tag*

CGTP

`CgtpMasterMBean`
`nhas-object=cgtp_stats`

```
CgtpMBean
  nhas-object=cgtp_stats,node=NodeID

CgtpFilterMBean
  nhas-object=cgtp_stats,node=NodeID,cgtp=filtering

CgtpReliableLinkStatisticsMBean
  nhas-object=cgtp_stats,node=NodeID,alias=AliasNumber0,address=IPaddress
```

Index

D

E

F

H

I

J

M

N

P

R

S

T

U

V

W