NORTEL NETWORKS

Preside Multiservice Data Manager

Alarm and Status API

Reference Guide

241-6001-203

Preside Multiservice Data Manager

Alarm and Status API

Reference Guide

Publication: 241-6001-203
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003

Printed in Canada

Publication history

December 2003

14.3RSUP Standard
Commercial availability

Contents

About this document

The following topics are discussed in this section:

Who should read this document and why

This document is for those who write custom applications for Preside Multiservice Data Manager (MDM) using the Alarm and Status Application Programming Interface (API). This document describes the Alarm and Status API and how to use it.

What you need to know

This document assumes knowledge of the following

- how to log on to an Preside Multiservice Data Manager (MDM) workstation
- the UNIX operating system
- the network model
- the Application Programming Interface

How this document is organized

241-6001-203 *Preside MDM Alarm and Status API Reference Guide* contains the following sections:

- "Introduction" (page 15) describes the managed objects, API messages and the event model of the Alarm and Status API.
- "Object model" (page 23) describes what objects are managed in the Alarm and Status API.
- "Using the Alarm and Status API" (page 61) describes how to use the Alarm and Status API.
- "Compliance statement" (page 87) details how the Alarm and Status API complies with the functionality described in the 241-6001-200 *Preside MDM Application Programming Interface Primer*.
- "Inbound Alarm API" (page 99) describes the specialized capabilities of the Alarm Status API and GMDR to support the direct injection of external network management data from external devices. The IMDR API is the recommended replacement for this API.
- "IMDR API" (page 113) describes the specialized capabilities of the IMDR API to support the direct injection of surveillance data from external devices, which are not supported by
- Preside Multiservice Data Manager (MDM).

What's new in this document

There are no changes in this document for this release.

Text conventions

This document uses the following text conventions:

- `nonproportional spaced plain type`

  Nonproportional spaced plain type represents system generated text or text that appears on your screen.

- **nonproportional spaced bold type**

  Nonproportional spaced bold type represents words that you should type or that you should select on the screen.

- *italics*

  Statements that appear in italics in a procedure explain the results of a particular step and appear immediately following the step.

  Words that appear in italics in text are for naming.

- `[optional_parameter]`

  Words in square brackets represent optional parameters. The command can be entered with or without the words in the square brackets.

- `<general_term>`

  Words in angle brackets represent variables which are to be replaced with specific values.

- UPPERCASE,lowercase

  In Preside Multiservice Data Manager (MDM), uppercase and lowercase letters that appear in UNIX commands and parameters must be matched exactly. The system matches upper and lowercase characters differently.

- |

  This symbol separates items from which you may select one; for example, ON|OFF indicates that you may specify ON or OFF. If you do not make a choice, a default ON is assumed.

- ...

  Three dots in a command indicate that the parameter may be repeated more than once in succession.

The term absolute pathname refers to the full specification of a path starting from the root directory. Absolute pathnames always begin with the slash (/) symbol. A relative pathname takes the current directory as its starting point, and starts with any alphanumeric character (other than /).

Related documents

See the following documents for related information:

- 241-6001-011 *Preside MDM Fault Management User Guide*
- 241-6001-118 *Preside MDM SNMP Surveillance Adapter Guide*
- 241-6001-200 *Preside MDM Application Programming Interface Primer*
- 241-6001-303 *Preside MDM Administrator Guide*

Chapter 1
Introduction

This section describes the purpose of the Alarm and Status Application Programming Interface (API). It also defines the object classes, action types, sieves, event reports, and API messages used by this API.

Alarm and Status API

The Alarm and Status API is an ASCII interface that provides access to the following information:

- recent Preside Multiservice Data Manager (MDM) alarms (the alarm information is presented in a format that is common to both DPN and Passport)
- current raw state of the DPN and Passport components
- notification of alarm, status, and rawStateChange events

The Alarm and Status API is one of many MDM workstation APIs. The MDM workstation custom programming environment is shown in the figure "MDM workstation custom programming environment" (page 17). The common behavior and rules for all MDM APIs are described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

The data collection daemon (DCD) server has specific API requests. The DCD API is similar to the Alarm and Status API. For more information, see the section on DCD API requests in 241-6001-118 *Preside MDM SNMP Surveillance Adapter Guide*.

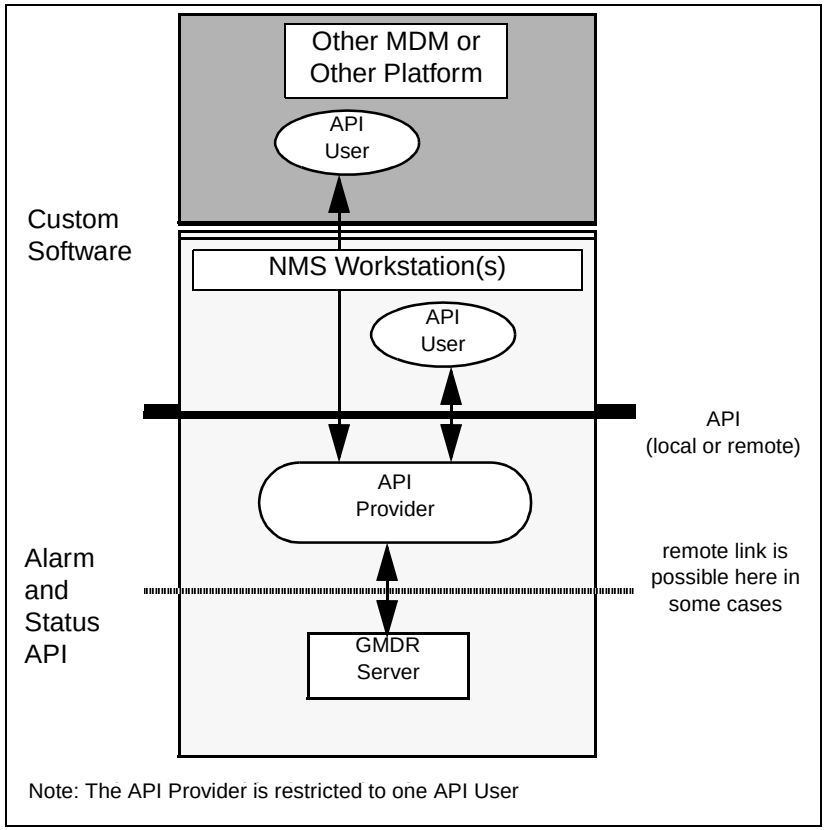
The Alarm and Status data is accessed through the General Management Data Router (GMDR) Server, which finds the data and delivers it to the Alarm and Status API. The Alarm and Status API consists of two functional elements: the API Provider, and the GMDR server.

The API Provider is the software that provides access to MDM data. The API User is the script, program, or human user that communicates with the API Provider over the interface. The API is defined in terms of the messages passed between the API User and the API Provider.

The GMDR server uses the services of the Passport Management Data Router (FMDR) and the DPN Management Data Router (DMDR). The FMDR sends the initial status of Passport components, and alarm and state change event notifications from Passport components to the GMDR. Similarly, the DMDR provides surveillance information about DPN devices. For more information on the servers, see the section on API messages in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

In a typical session between an API Provider and an API User, the API User issues requests and receives responses and/or errors from those requests. In addition, the API Provider may send event reports to the API User. The API Provider also sends a version message during initialization, and an end message during termination. The API sends an end message to request termination.

Figure 1
MDM workstation custom programming environment



Object classes

The Alarm and Status API has the following classes of objects:

- node
- link
- network
- alarmRecord
- log
- sieve

Refer to “Object classes” (page 24) for a description of the managed objects.

Action types

The Alarm and Status API has no action types.

Sieves and event reports

The sieve objects of the Alarm and Status API control the flow of notifications to the API User.

The event reports of the Alarm and Status API are:

alarm
indicates a failure in the network or the correction of a previous failure

rawStateChange
indicates a changed value for a raw state attribute of a node managed object

ackStateChange
indicates a changed value for a ack state attribute of a node managed object

status
provides a periodic report of statistical data

API messages

In a session between the API Provider and the API User, the API User issues requests and receives responses (and where appropriate, error messages) from those requests.

> ***Note:*** After sieve creation, the API Provider may send event reports to the API User.

Instead of a successful response message, the API Provider may return an error message to indicate some failure in the processing of a request. After sending an error message, the API Provider stops processing the request and sends an end-of-response message.

The end-of-response message indicates that no more responses are forthcoming from the API Provider and that the request processing is complete. For the Alarm and Status API, the message types are:

- REGISTER
  The use of the REGISTER message is mandatory; no commands are accepted from the API User until the REGISTER message is received. No password is required for registration. The *userId* attribute is mandatory.
  It is recommended that the *userId* attribute is formatted as *APIuser*, where *user* is specific to the API User. The *userId* attribute is used by the GMDR Admin Tool to identify which API Users are using the GMDR services. Refer to *241-6001-303 Preside MDM Administrator Guide*, for information.

- DEREGISTER
  The DEREGISTER message is not mandatory; the API User could just terminate an API session (refer to “Terminating Alarm and Status API access” (page 67)). After a DEREGISTER message, no commands are accepted other than the REGISTER command.

- CREATE
  The CREATE message is only supported for the sieve object.The *admState* defaults to 1 if the admState attribute is not supplied. You must specify one *eventType* of the *eventFilter* attribute. The *obj_id*:, *sup_obj_id*:, and *ref_obj_id*: message lines are not supported. Alarm-reporting sieves support the special Boolean parameter *explicitClears*. This parameter issues an explicit proxy clear notification for each SET alarm being cleared.

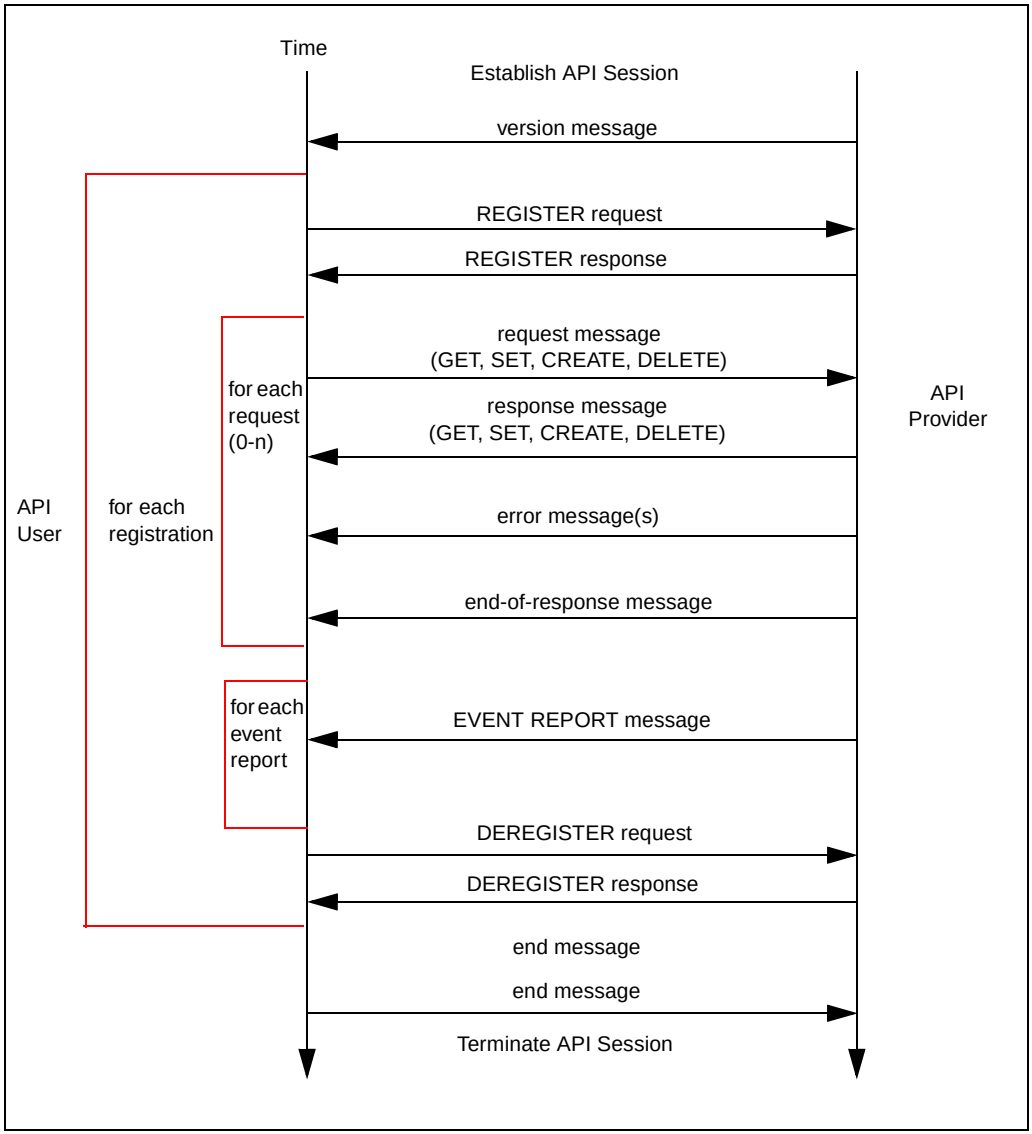
- DELETE
  The DELETE message is only supported for the sieve object. Scoping and filtering are not supported. Access to the sieve information is restricted to the sieve creators.

- GET
  The GET message is supported on every object except the root object. The root object is not addressable and thus cannot be used as a basis for a scoped or filtered request.
  Access to the sieve information is restricted to the sieve creators. The filters are subject to the filter restrictions documented in “Object model” (page 23).

- SET
  The SET message is only supported for the sieve object. Scoping and filtering is not supported. The ADD, DEL and DEF modification types are not supported.
- EVENT REPORT
  Only one type of event report may be specified for each sieve. The *eventType* of the *eventFilter* attribute is required. The *repFilter* attribute is supported only for the *compId* and the *objectClass* attributes. The *repInfo sieve* attribute is not supported.
  Access to the sieve information is restricted to the sieve creators. The filters are subject to the filter restrictions documented in “Object model” (page 23).

The figure “Time line diagram” (page 21) shows an overview of the session events. It represents the normal sequence of messages between the API Provider and the API User. For more details on the message types, refer to the section on API messages in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 2
Time line diagram



Chapter 2
Object model

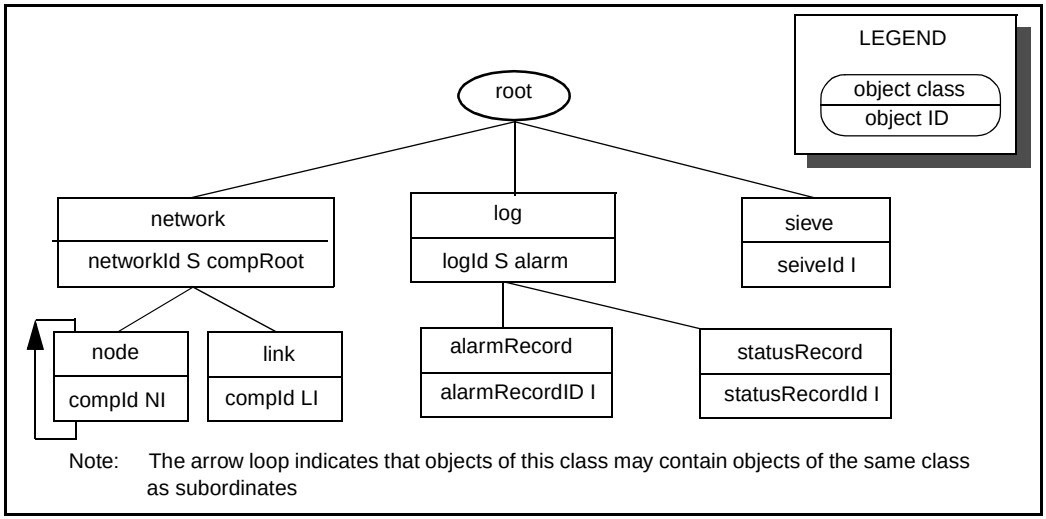
This section describes the containment hierarchies, object classes, and object class definitions used by this API.

This section contains the following information:

Containment hierarchies

The containment hierarchy is a set of managed objects that are visible through the API and are arranged in a single structure with a single root. The name of a managed object is based on its position in the hierarchy, and is represented as an attribute of the managed object (the naming attribute). The Alarm and Status API object model naming hierarchy is shown in the figure "Alarm and Status API object model naming hierarchy" (page 24).

Figure 3
Alarm and Status API object model naming hierarchy



Object classes

The Alarm and Status API provides alarm and status information to API Users through the following classes of objects:

- node
- link
- network
- alarmRecord
- log
- sieve

node

The node object represents Preside MDM managed components (Passport, DPN, or other types of components obtained from the SNMP Surveillance Adapter or injected by customer scripts). The node object emits the following event notifications: alarms, rawStateChange, ackStateChange and status.

link

The link object represents various trunks and links between Preside Multiservice Data Manager (MDM) managed components. The link object emits alarms.

network

The network object represents the root of the node and link hierarchy. There is only one object instance named *compRoot*.

alarmRecord

The alarmRecord object represents a logged alarm notification in a format common to both DPN and Passport. The number of alarmRecords stored is specified by a GMDR command line option. When the maximum number of stored alarms is obtained, and a new alarm is received, the oldest non-active alarm is discarded. If the alarm log only contains active alarms, the oldest alarm that has a lower severity than the new alarm is discarded. To prevent a component with many alarms from filling up the log, the maximum number of alarms stored for one component can be specified by a GMDR command line option.

statusRecord

The statusRecord object represents DPN-100 status record types.

log

The log object represents a repository that stores *logged* notifications. There are two log object instances: *alarm* that stores alarm notifications, and *status* that stores status notifications.

sieve

The sieve object controls the flow of event reports to the API Users.

Alarm clearing

Alarm clearing can occur explicitly by an alarm self-clearing process or implicitly by model administrative actions or events.

Explicit alarm self clearing occurs when an alarm overrides, or clears, any previous alarm from the same module. Although from the same module, the clearing alarm might not be from the same subcomponent or have the same fault code as the previous alarm. Self clearing alarms support asynchronous alarm notifications.

Alarms to be overridden are normally indicated in the clearing alarm by the correlatedNotifications attribute. The value of the correlatedNotifications attributes match the notificationId attributes of outstanding active alarms for the same module. The notificationId remains unique within a given module until a reset occurs.

Implicit alarm clearing can occur as a result of administrative actions, for example when the GMDR database is reset. Implicit alarm clearing can also occur as a result of events, for example when there is a loss of connection to an OA or to a server. The only notification from this loss of connection is a rawStateChange notification to UNKNOWN on the lost module(s). In such cases, a single alarm clears a potentially large number of outstanding alarms. Note that SET alarms can override other SET alarms but typically change only the severity of the reported problem.

Identification of cleared alarms can be difficult. For example, it is difficult to identify cleared alarms if you use a relational database to maintain an active alarm list. Locating cleared notificationIds implies a hierarchical component name search in the stored alarms. To overcome the difficulty in identifying cleared alarms, the Alarm and Status API can provide explicit notification of each cleared alarm.

You can request explicit notification of cleared alarms by using the explicitClears parameter on the alarm sieve creation query. When you use this parameter, the sieve reports the usual alarms and also includes the explicit proxy clear notifications. These notifications also cover alarms cleared by administrative actions and events. Since the output can represent a potentially large number of notifications, the explicitClear notifications are not reported by default. In addition, the notifications contain only the minimal amount of information you need to identify the cleared alarm. Special values of attributes originatorClass (clear) and commentData (Explicited CLEAR Alarm generated by GMDR) distinguish between cleared alarms and an explicitly cleared alarms.

Enabling Explicit Clear Notifications causes the sieves in this mode to forward the Active Set alarms from GMDR's resynchronization with its sub-servers. Normally, these alarms are not emitted by the sieves to avoid sending duplicate information. Forwarding this information maintains an Active Alarm list that is synchronizationed with GMDR.

Object class definitions

The following tables present the attribute makeup of each object type and the notifications being emitted by these objects. The table "Attribute list and notification list table format" (page 29) describes the format of the subsequent attribute and notification tables.

The attribute part of the table consists of five columns that include the following information:

- name of the attribute
- type of attribute value
- valid operators for filtering
- attribute significance
- information about whether the attribute is mandatory (M), optional (O), an alarm event (A), a status event (S), or present in explicit proxy clear notifications (E)

The notification part of the table consists of four columns that include the following information:

- name of the event type,
- event information attributes that are conveyed as part of the notification
- a brief description of the cause of the notification
- information about whether the attribute is mandatory (M), optional (O), an alarm event (A), or a status event (S)

Tables 2 to 6 define the attributes and notifications of the object classes of the Alarm and Status API.

The tables from 2 to 6 are the following:

- “Node class attributes and notifications” (page 30)
- “Node class attributes and notifications” (page 30)
- “Network class attributes” (page 33)
- “Log class attributes” (page 34)
- “AlarmRecord class attributes” (page 34)

Tables 7 to 9 define the event notifications. For a complete description of the sieve type, refer to 241-6001-200 *Preside MDM Application Programming Interface Primer*.

The tables 7 to 9 are the following:

- “Mapping of NCS action to alarmType attribute” (page 49)
- “Mapping of NCS severity and type to severity” (page 50)
- “Mapping of NCS action to probable cause” (page 50)

Table 1
Attribute list and notification list table format

Attributes				
Attribute	**Data type**	**Filter**	**Description**	**Notes**
Attribute name	Type of attribute value	Identifies the operators that can be used for filtering. Applies to GET requests and sieve filtering.	An explanation of attribute significance. The Description column indicates if the attribute is the Object ID naming attribute.	**M** - mandatory **O** - optional **A** - alarm event attribute **S** - status event attribute **E** - present in explicit proxy clear notifications
Notifications				
Event type	**Attributes**		**Description**	
Attribute name	The event information attributes		The notification cause	**M** - mandatory **O** - optional **A** - alarm event attribute **S** - status event attribute

Note: Mandatory means that the attribute is guaranteed to appear in the response. Optional means that the attribute may or may not appear in the response.

Table 2
Node class attributes and notifications

Node class attributes				
Attribute	**Data type**	**Filter**	**Description**	**Notes**
compCriticality	I	P,EQ, NE,GT, LT,GE, LE	Assigned Component Criticality value (from the Criticality Schema and Overrides configuration file). This value is in the 0-255 range.	M
compId	NI, LI	P, EQ, NE, LEFT, RIGHT, MIDDLE	The Object ID naming attribute. Object ID of the object for which an alarm is reported	M
objectClass	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Always node.	M
rawState	E	P, EQ, NE	State as received from GMDR **INSV** - in service; the component is functioning normally **ISTB** - in service troubled; the component is functioning but is experiencing some problem **OOS** - out of service; the component is not functioning **UNK** - unknown; the state of the component is not known and communications to the component is lost **NEX** - non-existent; the component is deleted from the Management Information Base (MIB) by an administrator	M

(Sheet 1 of 2)

Table 2 (continued)
Node class attributes and notifications

Node class attributes				
Attribute	**Data type**	**Filter**	**Description**	**Notes**
discoveryState	E	P, EQ, NE	This attribute exists for top level node components monitored by SMDR or FMDR, only. **discoveryUnknown** - the switch has not been polled or state walked **discoveryInProg** - a polling cycle or state walk has started but is not complete **discoveryDone** - one complete switch polling cycle or state walk has been completed.	M
Node class notifications				
Event type	**Attributes**		**Description**	**Notes**
alarm	refer to the table “AlarmRecord class attributes” (page 34)		refer to the table “AlarmRecord class attributes” (page 34)	M
rawState Change	refer to the table “Attributes of the rawStateChange event” (page 58)		refer to the table “Attributes of the rawStateChange event” (page 58)	M
ackState Change	refer to the table “Attributes of the rawStateChange event” (page 58)		refer to the table “Attributes of the rawStateChange event” (page 58)	M
status	refer to the table “Attributes of the status event (DPN only)” (page 51)		refer to the table “Attributes of the status event (DPN only)” (page 51)	M
(Sheet 2 of 2)				

Table 3
Link class attributes and notifications

Link class attributes				
Attribute	**Data type**	**Filter**	**Description**	**Notes**
compCriticality	I	P,EQ, NE,GT, LT,GE, LE	Assigned Component Criticality value (from the Criticality Schema and Overrides configuration file). This value is in the 0-255 range.	M
compId	NI, LI	P, EQ, NE, LEFT, RIGHT, MIDDLE	The Object ID naming attribute. Object ID of the object for which an alarm is reported	M
objectClass	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Always link	M
rawState	E	P, EQ, NE	State as received from GMDR **INSV** - in service; the component is functioning normally **ISTB** - in service troubled; the component is functioning but is experiencing some problem **OOS** - out of service; the component is not functioning **UNK** - unknown; the state of the component is not known and communications to the component is lost **NEX** - non-existent; the component is deleted from the Management Information Base (MIB) by an administrator	M
Link type notifications				
Event type	**Attributes**		**Description**	**Notes**

(Sheet 1 of 2)

Table 3 (continued)
Link class attributes and notifications

Link class attributes			
Attribute	**Data type** **Filter**	**Description**	**Notes**
alarm	refer to the table “AlarmRecord class attributes” (page 34)	refer to the table “AlarmRecord class attributes” (page 34)	M
rawState Change	refer to the table “Attributes of the rawStateChange event” (page 58)	refer to the table “Attributes of the rawStateChange event” (page 58)	M
(Sheet 2 of 2)			

Table 4
Network class attributes

Attribute	Data type	Filter	Description	Notes
objectClass	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Always network	M
networkId	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	The Object ID naming attribute. There is only one network object instance named *compRoot* representing the root of the node and link hierarchy.	M

Table 5
Log class attributes

Attribute	Data type	Filter	Description	Notes
logId	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	The Object ID naming attribute. There are two log object instances: *alarm* stores alarm notifications, and *log* stores status notifications.	M
objectClass	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Always log	M

Table 6
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
ackReason	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Reason for the acknowledgment or unacknowledgment. This attribute is always output in BLOCK format.	O,A
ackState	E	P,EQ, NE	Acknowledgment state of the alarm Possible values are: **notAcked** - alarm has never been acknowledged **acked** - alarm is currently acknowledged **unacked** - alarm is currently unacknowledged	O,A
ackTime	D	P,EQ, NE,GT, LT,GE, LE	Time when the alarm was last acknowledged or unacknowledged	O,A
(Sheet 1 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
ackUserId	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	A user identifier for the person who is associated with the alarm acknowledgment or unacknowledgment.	O,A
active	I	P,EQ, NE,GT, LT,GE, LE	For an alarmRecord, it indicates if the condition raised by a set alarm still exists. When a *set* alarm is logged, this attribute is 1. When a *clear* alarm is received, which indicates that the condition is corrected, this attribute is set to 0. For *message* and *clear* alarms, this attribute is always set to 0. For an alarm notification, this attribute is 1 for all *set* alarms, and 0 for all *message* and *clear* alarms.	M,A,E
administrative State (Passport only)	E	P,EQ, NE	Possible values are: **locked** - the component is administratively prohibited from performing services for its users **unlocked** - the component can perform services for its users **shuttingDown** - existing users of the component may still be serviced, but any new users are denied service. This attribute is derived from the Passport *administrativeState* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
alarmRecordId	I	P,EQ, NE,GT, LT,GE, LE	The Object ID naming attribute. Uniquely identifies an alarmRecord within the Alarm and Status MIB. (Not present in notifications.)	M

(Sheet 2 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
alarmStatus (Passport only)	SetOfE	P,EQ, NE	This attribute may have one or more of the following values: **underRepair** - the component is being repaired **alarmStatusCritical** - one or more critical alarms indicating a fault or failure are detected and are not cleared **alarmStatusMajor** - one or more major alarms indicating a fault or failure are detected and are not cleared **alarmStatusMinor** - one or more minor alarms indicating a fault or failure are detected and are not cleared **alarmOutstanding** - one or more alarms are detected and are not cleared **alarmEmpty** - the empty set This attribute is derived from the Passport *alarmStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
alarmType	E	P, EQ, NE	General explanation of why the alarm was generated Possible values are: communications, qualityOfService, processing, equipment, environmental, security, debug, operator, and unknown. This attribute is derived from the Passport *alarmType* attribute for Passport-originated alarms, and derived from the NCS *action* field according to the table “Mapping of NCS action to alarmType attribute” (page 49) for NCS-originated alarms.	M,A

(Sheet 3 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
availabilityStatus (Passport only)	SetOfE	P,EQ, NE	This attribute may have one or more of the following values: **offLine** - the component requires a routine operation to place it on-line and make it available **offDuty** - the component is made inactive by an internal control process in accordance with a time schedule; under normal conditions, it should come back into operation through the same mechanism **dependency** - the component cannot operate because some other component on which it depends is unavailable **degraded** - the service provided by the component is degraded in some respect, such as speed or operating capacity; the service remains available **notInstalled** - the component is not present **logFull** - indicates a log full condition **inTest** - the component is undergoing a test procedure **failed** - the component has an internal fault that prevents it from operating **powerOff** - the component requires power **procEmpty** - the empty set This attribute is derived from the Passport *availabilityStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
commentData	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Description of the problem This attribute is always output in BLOCK format. It is derived from the Passport *commentData* attribute for Passport-originated alarms, and from the NCS *comment text* field for NCS-originated alarms.	O,A,E

(Sheet 4 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
compCriticality	I	P,EQ, NE,GT, LT,GE, LE	Assigned Component Criticality value (from the Criticality Schema and Overrides configuration file). This value is in the 0-255 range.	M,E
compId	NI, LI	P, EQ, NE, LEFT, RIGHT, MIDDLE	The Object ID naming attribute. Object ID of the object for which an alarm is reported	M,E
controlStatus (Passport only)	SetOfE	P,EQ, NE	This attribute may have zero or more of the following values: **subjectToTest** - the component is available to normal users, but tests may be conducted simultaneously, causing it to show unusual behavior to users **partOfServicesLocked** - a manager has restricted a particular part of the service from the users **reservedForTest** - the component is administratively unavailable because it is undergoing a test procedure **suspended** - service is administratively suspended to the users **controlEmpty** - the empty set This attribute is derived from the Passport *controlStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
(Sheet 5 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
correlated Notifications	SetOfI	P,EQ, NE,GT, LT,GE, LE	Identifies alarms related to this alarm For a clear alarm indicates the notification IDs of the set alarms that are cleared. For a set alarm, indicates the notification ID of the set alarm which it replaces (for example, where the severity of an alarm changes, no clear alarm is generated, only two consecutive set alarms). This attribute is generated by the FMDR server for Passport-originated alarms, and by the DMDR server for DPN-originated alarms.	O,A
customerId	I	P,EQ, NE,GT, LT,GE, LE	Customer identification	M,A,E
event	E	P,EQ, NE	Possible values are: **message** - the alarm should be recorded, but no associated clear is sent; transient events or alarms that just provide information fall into this category. **set** - the alarm should be recorded; a clear follows when the condition is corrected. **clear** - the condition is corrected. This attribute is derived from the Passport *activeListStatus* attribute for Passport-originated alarms, and derived from the NCS *type* field for NCS-originated alarms.	M,A,E
expertData	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Additional information that an expert may use if the alarm cannot be resolved by the operator This attribute is always output in BLOCK format. It is derived from the Passport *internalData* attribute for Passport-originated alarms, and from the NCS *expert data* field for NCS-originated alarms.	O,A

(Sheet 6 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
faultCode	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	8-digit code to further specify the alarm The first four digits identify the source of the alarm, and the last four digits identify the alarm. The fault code can be used as an index into the alarm document which describes the recommended corrective action. This attribute is derived from the Passport *ntpIndex* attribute for Passport-originated alarms, and from the NCS *condition* field for NCS-originated alarms.	M,A,E
fileLineNumber (Passport only)	I	P,EQ, NE,GT, LT,GE, LE	Additional information for an expert or internal support This attribute is derived from the Passport *fileLineNumber* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
fileName (Passport only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Additional information for an expert or internal support This attribute is derived from the Passport *file* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
fileVersion (Passport only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Additional information for an expert or internal support This attribute is derived from the Passport *fileVersion* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A

(Sheet 7 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
ncsAction (DPN only)	E	P,EQ, NE	Describes the functional area that most likely caused the alarm Possible values are: ncsServiceData, ncsHardware, ncsSoftware, ncsSecurity, ncsProtocol, ncsDebug, ncsNetwork, ncsEngineering, ncsOperations, ncsUnclassified This attribute is not present in Passport-originated alarms.	O,A
ncsCondition Mnem (DPN only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Provides a general indication of the meaning of the alarm Possible values are: TRAPDATA, ACTIVATE, INVALID, MISSING, DUPLICAT, MEMORY, CONGEST, FAILED, REFUSED, TIME_OUT, CRITICAL, OOS, THRESHLD, ENABLED, DISABLED, PROBE, CALL_DOWN, CALL_BLK, and DISCARD This attribute is not present in Passport-originated alarms.	O,A
ncsDeviceName (DPN only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Device name This attribute is not present in Passport-originated alarms.	O,A
ncsDeviceType (DPN only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Device type This attribute is not present in Passport-originated alarms.	O,A
ncsNamsId (DPN only)	I	P,EQ, NE,GT, LT,GE, LE	Uniquely identifies a DPN module This attribute is not present in Passport-originated alarms.	O,A
(Sheet 8 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
ncsSeqNum (DPN only)	I	P,EQ, NE,GT, LT,GE, LE	Alarm sequence number This attribute is not present in Passport-originated alarms.	O,A
ncsSeverity (DPN only)	E	P,EQ, NE	Severity of the alarm Possible values are: **ncsUnknown** - no (or unknown) action is required **ncsDegrade** - no action is required (normally both a set and a clear are generated); operation of the resource has been temporarily degraded **ncsOverload** - no action is required (normally both a set and a clear are generated); the component is experiencing an overload condition and the customer or network data has been destroyed or removed because of a lack of capacity in a component **ncsMinor** - urgent corrective action should be taken to correct a problem; service to one customer is affected **ncsMajor** - immediate corrective action is required; service to more than one customer is affected **ncsWildcard** - all alarms for this component are cleared This attribute is not present in Passport-originated alarms.	O,A

(Sheet 9 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
notificationId	I	P,EQ, NE,GT, LT,GE, LE	Alarm identifier provided by the generating device The identifier is unique within a device; it may be used by the *correlatedNotification* attribute of subsequent alarms. This attribute is derived from the Passport *notificationId* attribute for Passport-originated alarms, and derived from the NCS *record sequence number* field for NCS-originated alarms.	M,A,E
objectClass	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Always alarmRecord (Not present in notifications.)	M
operationalState (Passport only)	E	P,EQ, NE	Possible values are: **enabled** - the component is partially or fully operable and available for use **disabled** - the component is totally inoperable and unable to provide service This attribute is derived from the Passport *operationalState* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
operatorData	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Additional information that an operator may use This attribute is always output in BLOCK format. It is derived from the Passport *operatorData* attribute for Passport-originated alarms, and from the NCS *operator data* field for NCS-originated alarms.	O,A
originatorClass	E	P, EQ, NE	Indicates the origin of the alarm Possible values include dpn, passport, external, snmp, and clear. The special value *clear* is also used to distinguish explicit proxy clear alarms.	M,A,E

(Sheet 10 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
probableCause	E	P, EQ, NE	Standardized cause for the event - Possible values are: lossOfSignal, lossOf Frame, framingError, localTransmission Error, remoteTransmissionError, callEstablishment Error, degradedSignal, commSubsystem Failure, commProtocolError, lanError, dteDceInterfaceError, responseTimeExcessive, queueSizeExceeded, bandwidthReduced, retransmissionRateReduced, thresholdCrossed, performanceDegraded, congestion, atOrNearCapacity, storageCapacityProblem, versionMismatch, corruptData, cpuCyclesLimitExceeded, softwareError, softwareProgramError, softwareProgramTermination, fileError, outOfMemory, underlyingResourceUnavailable, applicationSubsystemFailure, configurationError, powerProblem, timingProblem, processorProblem, dataSetModemError, multiplexorProblem, transmitterFailure, receiverFailure, outputDeviceError, inpuDeviceError, ioDeviceError, equipmentFailure, adapterError, duplicateInfo, infoMissing, infoModification, infoOutOfSequence, unexpectedInfo, denialOfService, outOfService, proceduralError, otherOperational, cableTamper, intrusionDetection, otherPhysical, authenticationFailure, breachOfConfidence, nonRepudiationFailure, unauthorizedAccess, otherSecurityService, delayedInfo, keyExpired, outOfHoursActivity, operationalCondition, debugging, probCauseUnknown, activeVirtualCircuit, networkServerIntervention, inactiveVirtualCircuit, enclosureDoorOpen, excessiveVibration, (...continued)	M,A

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
			fireDetected, floodDetected, heatVentCoolingProblem, humidityUnacceptable, leakDetected, materialSupplyExhausted, pressureUnacceptable, pumpFailure, temperatureUnacceptable, and toxicLeakDetected	
			This attribute is derived from the Passport *probableCause* attribute for Passport-originated alarms, and from the NCS *action* field according to the table “Mapping of NCS action to probable cause” (page 50) for NCS-originated alarms.	O,A
proceduralStatus (Passport only)	SetOfE	P,EQ, NE	This attribute may have one or more of the following values: **initializationRequired** - the component requires initialization before it can perform normal functions; the operator initiates it **notInitialized** - the component requires initialization before it can perform normal functions; the component initializes itself autonomously **initializing** - the component is being initialized **reporting** - the component has completed some operation and is notifying the results (for example, a test process) **terminating** - the component is in the termination phase **procEmpty** - the empty set This attribute is derived from the Passport *proceduralStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	

(Sheet 12 of 16)

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
processId (Passport only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Additional information for an expert or internal support This attribute is derived from the Passport *pid* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
rawState	E	P, EQ, NE	Possible values are: **INSV** - in service; the component is functioning normally **ISTB** - in service troubled; the component is functioning but is experiencing some problem **OOS** - out of service; the component is not functioning **UNK** - unknown; the state of the component is not known and communications to the component is lost **NEX** - non-existent; the component is deleted from the Management Information Base (MIB) by an administrator This attribute is computed by the Passport Management Data Router (FMDR) server for Passport, by the DPN Management Data Router (DMDR) server for the DPN, and does not appear in NCS-originated alarms.	O,A
related Components (Passport only)	SetOf NI	P, EQ, NE, LEFT, RIGHT, MIDDLE	Components related to the alarmed component; usually the supporting hardware that the affected component resides on This attribute is derived from the Passport *relatedComponentName* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
reporterName (DPN only)	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Name of the Operations Agent (OA) from which the alarm was received This attribute is not present in Passport-originated alarms.	O,A
(Sheet 13 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
severity	E	P, EQ, NE	Severity of the alarm Possible values are: **indeterminate** - severity level cannot be determined **critical** - immediate corrective action is required; the resource is completely disabled and service is affected **major** - urgent corrective action is required; the resource is severely disabled and service is affected **minor** - corrective action should be taken to prevent a more serious fault; the resource is partially disabled, but service is not affected **warning** - action should be taken to diagnose and correct the problem; some problem has been detected, but the resource is not disabled and service is not affected **cleared** - all alarms with a notificationId matching this alarm's correlatedNotifications are cleared This attribute is derived from the Passport *severity attribute* for Passport-originated alarms, and from the NCS *severity* and *type* fields according to the table "Mapping of NCS severity and type to severity" (page 50) for NCS-originated alarms.	M,A,E
(Sheet 14 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
standbyStatus (Passport only)	E	P,EQ, NE	This attribute may have one of the following values: **hot** - the component is not providing service, but is operating in synch mode to take over immediately **cold** - the component is not providing service and is not synchronized; switching to this component requires some intervention **providing** - the component is providing service **standbyEmpty** - the empty set This attribute is derived from the Passport *standbyStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
time	D	P,EQ, NE,GT, LT,GE, LE	Time when the alarm or status was generated	M,A,E
(Sheet 15 of 16)				

Table 6 (continued)
AlarmRecord class attributes

Attribute	Data type	Filter	Description	Notes
unknownStatus (Passport only)	E	P,EQ, NE	Indicates the state of the component is unknown When the value of the attribute is true, the value of the state attributes may not reflect the actual state of the resource. This attribute is derived from the Passport *unknownStatus* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
usageState (Passport only)	E	P,EQ, NE	Possible values are: **usageIdle** - the component is not currently in use **usageActive** - the component is in use, and has sufficient spare capacity to provide for additional users **usageBusy** - the component is in use, but it has no spare operating capacity to provide for additional users This attribute is derived from the Passport *usageState* attribute for Passport-originated alarms, and does not appear in NCS-originated alarms.	O,A
(Sheet 16 of 16)				

Table 7
Mapping of NCS action to alarmType attribute

Action	alarmType
serviceData	processing
hardware	equipment
software	processing
security	security
protocol	communication
(Sheet 1 of 2)	

Table 7 (continued)
Mapping of NCS action to alarmType attribute

Action	alarmType
debug	debug
network	communication
engineering	processing
operation	operator
unclassified	unknown
(Sheet 2 of 2)	

Table 8
Mapping of NCS severity and type to severity

NCS severity (and type)	Severity
any value (clear)	cleared
unknown (message or set)	indeterminate
degrade (message or set)	warning
overload (message or set)	warning
minor (message or set)	major
major (message or set)	critical
wildcard (any value)	cleared

Note: If the NCS value of type is clear, then the severity value is cleared regardless of the value of the NCS severity. If the NCS value of severity is wildcard, the severity value is cleared regardless of the value of the NCS type.

Table 9
Mapping of NCS action to probable cause

Action	probableCause
service data	configurationError
hardware	equipmentFailure
software	softwareError
security	otherSecurityService
protocol	communicationProtocol
debug	debugging
network	communicationsSubsystemFailure
engineering	underlyingResourceUnavailable
operations	operationalCondition
unclassified	probCauseUnknown

Table 10
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
active	I	EQ, NE, P	In a status notification, the value of the active attribute is always 1.	M,S
reporterName	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Name of the Operations Agent (OA) from which the status was received This attribute is not present in Passport-originated alarms.	M,S
statusType	E	EQ, NE	Specifies the type of status change notification Possible values are: commonSubsystem, processor, trunk, networkLink, x25Gateway, x75Gateway, diu, and diuSubsystem	M,S

(Sheet 1 of 8)

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
Common subsystem status type attributes				
avgFreeQueue	I	P,EQ, NE,GT, LT,GE, LE	Common average free queue percentage	O,S
busUtilization	I	P,EQ, NE,GT, LT,GE, LE	Common bus utilization percentage	O,S
freeQueue	I	P,EQ, NE,GT, LT,GE, LE	Common free queue percentage	O,S
minFreeQueue	I	P,EQ, NE,GT, LT,GE, LE	Common minimum free queue percentage	O,S
DIU status type attributes				
diuActiveSPMStatus	I	P,EQ, NE,GT, LT,GE, LE	DIU-active SPM status 0 = down, 1 = up	O,S
diuStandbySPMStatus	I	P,EQ, NE,GT, LT,GE, LE	DIU-standby SPM status 0 = down, 1 = up	O,S
diuType	I	P,EQ, NE,GT, LT,GE, LE	DIU type 0 = single SPM DIU, 1 = dual SPM DIU	O,S
(Sheet 2 of 8)				

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
DIU subsystem status type attributes				
subdiuNumOfConnection	I	P,EQ, NE,GT, LT,GE, LE	DIU subsystem - total number of connections in the subsystem	O,S
subdiuSubsystemStatus	I	P,EQ, NE,GT, LT,GE, LE	DIU subsystem status 0 = down, 1 = up	O,S
Network link status type attributes				
netLinkLinkUtilization	I	P,EQ, NE,GT, LT,GE, LE	netLink link utilization percentage	O,S
netLinkLocalLinkMnem	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	netLink local link mnemonic	O,S
netLinkRemoteCompId	NI	P, EQ, NE, LEFT, RIGHT, MIDDLE	netLink remote component identifier	O,S
netLinkType	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	netLink type Possible values are: **NL** - Network Link **DBNL** - Dial Backup Network Link **DNL** - Dial Network Link **DINL** - Dial In Link **FRNL** - Frame Relay Network Link **MPANL** - MPA Network Link	O,S

(Sheet 3 of 8)

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
Processor status type attributes				
proAvgFreeQueue	I	P,EQ, NE,GT, LT,GE, LE	Processor average free queue percentage	O,S
proCpuUtilization	I	P,EQ, NE,GT, LT,GE, LE	Processor CPU utilization percentage	O,S
proFreeHeap	I	P,EQ, NE,GT, LT,GE, LE	Processor free heap percentage	O,S
proFreeQueue	I	P,EQ, NE,GT, LT,GE, LE	Processor free queue percentage	O,S
proMinFreeQueue	I	P,EQ, NE,GT, LT,GE, LE	Processor minimum free queue percentage	O,S
proPPSReceived	I	P,EQ, NE,GT, LT,GE, LE	Processor packets/second, received	O,S
proPPSTransmitted	I	P,EQ, NE,GT, LT,GE, LE	Processor packets/second, transmitted	O,S

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
proServiceName	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Processor service name	O,S
proType	I	P,EQ, NE,GT, LT,GE, LE	Processor type	O,S
Trunk status type attributes				
trunkLinkUtilization	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Trunk link utilization percentage	O,S
trunkLocalLinkMnem	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Trunk local link mnemonic	O,S
trunkRemoteCompId	NI	P, EQ, NE, LEFT, RIGHT, MIDDLE	Trunk remote component identifier	O,S

(Sheet 5 of 8)

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
trunkLinkType	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	Trunk link Type. Possible values are: **TK** - Trunk **FRTK** - Frame Relay Trunk	O, S
X25 gateway status type attributes				
x25CpuUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.25 CPU utilization percentage	O,S
x25GatewayId	I	P,EQ, NE,GT, LT,GE, LE	X.25 gateway identifier	O,S
x25LcnUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.25 logical channel utilization percentage	O,S
x25LinkUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.25 link utilization percentage	O,S
x25NIConfig	I	P,EQ, NE,GT, LT,GE, LE	X.25 network link configuration	O,S
x25PPSReceived	I	P,EQ, NE,GT, LT,GE, LE	X.25 packets/second, received	O,S

(Sheet 6 of 8)

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
x25PPSTransmitted	I	P,EQ, NE,GT, LT,GE, LE	X.25 packets/second, transmitted	O,S
x25RemoteGtyMnem	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	X.25 remote gateway mnemonic	O,S
X75 gateway status type attributes				
x75CpuUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.75 CPU utilization percentage	O,S
x75GatewayId	I	P,EQ, NE,GT, LT,GE, LE	X.75 gateway identifier	O,S
x75LcnUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.75 logical channel utilization percentage	O,S
x75LinkUtilization	I	P,EQ, NE,GT, LT,GE, LE	X.75 link utilization percentage	O,S
x75PacketsReceived	I	P,EQ, NE,GT, LT,GE, LE	X.75 packets/second, received	O,S

(Sheet 7 of 8)

Table 10 (continued)
Attributes of the status event (DPN only)

Attribute	Data type	Filter	Description	Notes
x75PacketsTransmitted	I	P,EQ, NE,GT, LT,GE, LE	X.75 packets/second, transmitted	O,S
x75RemoteGtyMnem	S	P, EQ, NE, LEFT, RIGHT, MIDDLE	X.75 remote gateway mnemonic	O,S
(Sheet 8 of 8)				

Table 11
Attributes of the rawStateChange event

Attributes	Description
compCriticality	Component Criticality for this node See the compCriticality attribute in the table “AlarmRecord class attributes” (page 34).
rawState	RawState for this node See the rawState attribute in the table “AlarmRecord class attributes” (page 34)
time	Time when the rawStateChange was generated

Table 12
Attributes of the ackStateChange event

Attributes	Description
compAckState	AckState for this component changed The component is either: notAcked or acked.
time	Time when the ackStateChange was generated

Chapter 3
Using the Alarm and Status API

This section explains how to install and use the Alarm and Status API.

This section contains the following information:

Code conventions

The following are the code conventions used in this document:

- \
  A backslash (\ ) indicates that the line of code is continued on the next line space.
- * or #
  A message line that starts with an asterisk (*) or an octothrope (#) is treated as a comment.
- The Alarm and Status API does not support quoted strings; enter strings without quotes.

Starting a session with the Alarm and Status API

When using the Alarm and Status API, the API User actually works with the process called the API Provider. To start a session with the Alarm and Status API, the API User must invoke the API Provider by entering the following:

```
/opt/MagellanNMS/bin/gmdrapi
```

If initialization fails, then one or more error messages are output, followed by an end message, and the termination of the API Provider.

If initialization succeeds, the response is similar to the following:

```
_version: x.y Alarm and Status API
```

where x and y are the major and minor version numbers.

Command line options

The API User may want to send the Alarm and Status API an input file containing many requests, and then send the responses into an output file. This is achieved by using the command line options.

```
/opt/MagellanNMS/bin/gmdrapi
[-o <output filename> | +o <output\
filename>] [-f <input file name>]\
[-width <output width> | -w <output\
width>] [-help] [-d] [-k] [-a] [-v]\
[-echo]
```

where:

`-o <output filename> | +o <output filename>` names a file to which the API User writes the query output. With the -o option, the new output overwrites the file if it already exists. With the +o option, the new output appends to the file. The default output filename is the standard output stream (*stdout*).

`-f <input file name>` names a file from which the API queries are read and then executed.

`-width <output width> | -w <output width>` sets the output line width. If the output is longer than the specified value, a new line (\ ) is inserted to respect the maximum line length. The default value is 255.

`-help` displays a help panel on the public options. If the -help option is used with this option, the help panel also describes the private options.

`-d` selects daemon mode. In daemon mode, the API Provider reads the API input and executes the queries until it reaches the final end-of-file. At that point, the API Provider continues processing any replies it receives. The only way to terminate the API Provider in daemon mode is to kill it or use control-C.

`-k` selects keyboard mode. In keyboard mode, the API Provider reads the API input for the file named by the -f option until it reaches the end-of-file; the then API Provider reads the standard input stream again.

`-a` selects asynchronous mode. In asynchronous mode, the API Provider reads and executes its API queries as soon as they are provided. This allows the execution of parallel queries. The alternate is synchronous mode, where the next query is not read until the current one is completed.

`-v` selects verbose mode. In verbose mode, the API Provider provides additional messages to indicate the processing on the standard error stream (*stderr*). All error messages received through the Internal Protocol Interface (IPI) are also echoed to *stderr*.

`-echo` selects echo mode, where all queries on the output stream are echoed as API comments.

Interactive session

If no command line arguments are provided, the API User is in an interactive session with the local host server and requests information that resides on the workstation. If you want to have an interactive session with a server that resides on another workstation, use the -h option.

> ***Note:*** An interactive session is not recommended. If an interactive session is used, use the -v option (verbose mode).

Example

```
/opt/MagellanNMS/bin/gmdrapi [-h <host name>] [-v]
```

where:

`-h <host name>` indicates the host name of the server to which the API Provider connects (by default, the API Provider always tries to connect to a local server).

`-v` selects verbose mode.

The response is:

```
8.0 Alarm and Status API copyright: 1991 - 1994 Nortel
Networks Corporation.
Connected to API Server "GMDR" on "bcars561".
_version: 8.0 Alarm and Status API
```

Input through an input file name

For this example, the file name is *toto*, and it contains the following command lines:

```
_cmd: REGISTER
_user_id: APIUser
<blank line>
_cmd: get
_obj_class: node
_obj_id: compId NI PM R73
_scope: base
_attr_id: all
<blank line>
```

Example

```
/opt/MagellanNMS/bin/gmdrapi/ -h <host name> -f toto
```

The response is:

```
_version: 8.0 Alarm and Status API
<blank line>
_user_id: APIUser
<blank line>
_end_resp: REGISTER
_time: 1994 02 07 15 15 33
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
_attr: rawState E INSV
_attr: compAckState E notAcked
_attr: compCriticality I 90
```

```
<blank line>
_end_resp: GET
_time: 1994 02 07 15 13 35
<blank line>
_end: USER_REQUEST
<blank line>
```

Input through piping

For this example, the same file *toto* is piped to the API Provider.

Example

```
cat toto | /opt/MagellanNMS/bin/gmdrapi -h <host name>
```

The response is:

```
_version: 8.0 Alarm and Status API
<blank line>
_user_id: APIUser
<blank line>
_end_resp: REGISTER
_time: 1994 02 07 15 15 33
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
_attr: rawState E INSV
_attr: compAckState E notAcked
_attr: compCriticality I 90
<blank line>
_end_resp: GET
_time: 1994 02 07 15 15 35
<blank line>
_end: USER_REQUEST
<blank line>
```

Alternately, the file *toto* can be piped to the API Provider as in the following example:

Example

```
/opt/MagellanNMS/bin/gmdrapi -h <host name> < toto
```

The response is:

```
_version: 8.0 Alarm and Status API
<blank line>
_user_id: APIUser
```

```
<blank line>
_end_resp: REGISTER
_time: 1994 02 07 15 15 39
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
_attr: rawState E INSV
_attr: compAckState E notAcked
_attr: compCriticality I 90
<blank line>
_end_resp: GET
_time: 1994 02 07 15 16 42
<blank line>
_end: USER_REQUEST
<blank line>
```

Program driven input

This method is similar to the piping method, except with program-driven input, the API queries are built by a program and fed to the API Provider interactively.

Syntax error in interactive session

When a syntax error occurs in interactive session, it is important that verbose mode is enabled. Verbose mode provides additional messages to indicate the processing of the error (which would otherwise be missing).

Example

```
_cmd: toto
```

The API Provider returns the following:

```
_error: SYNTAX_ERROR Invalid value in Command line
<blank line>
SYNTAX_ERROR Invalid value in Command line
<blank line>
_end_resp:
_time: 1994 02 07 15 35 11
<blank line>
The rest of this API query will be ignored.
Make sure it is terminated (blank line) before
entering\
a new one.
```

Terminating Alarm and Status API access

To terminate the Alarm and Status API access, use either an end-of-file (control-d) or issue the following request:

```
_end:
<blank line>
```

The response message is:

```
_end: USER_REQUEST
<blank line>
Exiting API Provider with 0 status.
```

Commands

Each command has a request message and a response, and/or error message(s).

Refer to the section on API messages in 241-6001-200 *Preside MDM Application Programming Interface Primer*, for more details on commands.

The Alarm and Status API supports the following commands:

- REGISTER
- DEREGISTER
- GET
- CREATE sieves and event reports
- SET
- DELETE

The following sections describe the Alarm and Status API support of these commands.

The REGISTER command

The REGISTER command must be the first command from the API User. No password is required for registration.

Using the REGISTER command

The following REGISTER request registers the Alarm and Status API with the *userId* attribute value of *APItest*.

```
_cmd: REGISTER
_user_id: APItest
<blank line>
```

If the REGISTER request message is successful, the response message is:

```
_user_id: APItest
<blank line>
```

followed by an end-of-response message:

```
_end_resp: REGISTER
_time: 1994 01 31 09 46 53
<blank line>
```

The DEREGISTER command

The DEREGISTER command is optional. After a DEREGISTER message, no commands are accepted other than the REGISTER command.

Using the DEREGISTER command

To deregister, issue the following request message:

```
_cmd: DEREGISTER
<blank line>
```

If the DEREGISTER request is successful, the response message is:

```
<blank line>
_end_resp: DEREGISTER
_time: 1994 01 31 09 46 53
<blank line>
```

The GET command

The GET request retrieves values of attributes of all the Alarm and Status API objects: network, node, link, log, alarmRecord, statusRecord, and sieve objects. A GET request results in zero or more GET responses (one for each selected object), followed by an end-of-response message.

Getting network

To retrieve the rawState attribute of all nodes, issue the following request:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: ALL
_attr_id: rawState
<blank line>
```

If the GET request is successful, the typical response message is:

```
_obj_class: network
_obj_id: networkId S compRoot
<blank line>
_obj_class: node
_obj_id: compId NI EM NODE23
_attr: rawState E INSV
<blank line>
_obj_class: node
_obj_id: compId NI PM A6802 PRI 13
_attr: rawState E ISTB
<blank line>
...
_end_resp: GET
_time: 1994 01 31 10 05 06
<blank line>
```

To retrieve the compAckState attribute of all nodes, issue the following request:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: ALL
_attr_id: compAckState
<blank line>
```

If the GET request is successful, the typical response message is:

```
_obj_class: network
_obj_id: networkId S compRoot
<blank line>
_obj_class: node
_obj_id: compId NI EM NODE23
```

```
_attr: compAckState E notAcked
<blank line>
_obj_class: node
_obj_id: compId NI PM A6802 PRI 13
_attr: compAckState E acked
<blank line>
...
_end_resp: GET
_time: 1994 01 31 10 05 06
<blank line>
```

Getting node

To retrieve the subcomponents of EM NODEY TRK 77 (including EM NODEY TRK 77 itself) for which the value of the *rawState* attribute is OOS, issue the following request message:

```
_cmd: GET
_obj_class: node
_obj_id: compId NI EM NODEY TRK 77
_scope: ALL
_filter: rawState EQ E OOS
<blank line>
```

If the GET request is successful, the typical response message is:

```
_obj_class: node
_obj_id: compId NI EM NODEY TRK 77
<blank line>
_obj_class: node
_obj_id: compId NI EM NODEY2C5 TRK 77 L77
<blank line>
_obj_class: node
_obj_id: compId NI EM NODEY2C5 TRK 77 UNACKED $
<blank line>
_end_resp: GET
_time: 1994 01 31 12 25 49
<blank line>
```

Getting alarms

To retrieve all active alarms, issue the following request message:

Example

```
_cmd: GET
_obj_class: log
_obj_id: logId S alarm
```

```
_scope: ALL
_filter: active EQ I 1
_attr_id: ALL
<blank line>
```

If the GET request is successful, Passport and DPN alarms are received. A sample Passport alarm is:

```
_obj_class: alarmRecord
_obj_id: alarmRecordId I 28
_attr: time D 1994 01 31 17 54 47
_attr: compId NI EM NODE5 LP 7 X21 4
_attr: rawState E OOS
_attr: compCriticality I 40
_attr: alarmType E processing
_attr: faultCode S 70116501
_attr: severity E warning
_attr: probableCause E configurationError
_attr: notificationId I 117440514
_attr: originatorClass E passport
_attr: active I 1
_attr: customerId I 0
_attr: event E set
_block: _attr commentData S
Termination switch setting does not match provisioned.
Check switch setting or the provisioned data
(lineTerminationRequired).
_end_block:
_attr: fileName S PmsHwProcessHandler_Actor.cc
_attr: administrativeState E unlocked
_attr: operationalState E disabled
_attr: usageState E usageIdle
_attr: proceduralStatus E initializing
_attr: unknownStatus E unknownFalse
_attr: availabilityStatus E offline
_attr: standbyStatus E standbyEmpty
_attr: fileLineNumber I 451
_block: _attr fileVersion S
Oct 16 1993@ 00:28:47
_end_block:
_attr: processId S 7/0/22297
...
```

A sample DPN alarm is:

```
_obj_class: alarmRecord
_obj_id: alarmRecordId I 751
_attr: time D 1994 91 26 15 26 35
_attr: compId NI PM DINO-R59 NCS 6 AP 11
_attr: alarmType E operator
_attr: ncsDeviceType S RM
_attr: faultCode S 10054001
_attr: reporterName S TOPNCS
_attr: ncsConditionMnem S DISABLED
_attr: severity E major
_attr: probableCause E operationalCondition
_attr: notificationId I 227
_attr: originatorClass E dpn
_attr: active I 1
_attr: customerId I 0
_attr: event E set
_attr: ncsNamsId I 4059
_attr: ncsSeverity E ncsMinor
_block: _attr expertData S
EX: PROCESS ID:0B06 002C GLOBAL:008F SEQUENCE:0024
FFFF E000 FFFF E000 000A 0600 157C FFFF
FFFF 2606 FF0B 0D20 0FDB 0549 2020 FFFF
E000 FFFF E000 0060 F804 0B06 0028 0B06
002C 54FF FF69 0200 0000 0000 0000 0001
_end_block:
_attr: ncsAction E ncsOperations
_attr: ncsSeqNum I 227
_attr: ncsDeviceName S DINO-R59
...
<blank line>
_end_resp: GET
_time: 1994 01 31 10 05 06
<blank line>
```

Example

To retrieve all alarms for the EM NODE1 Passport node and all of its subcomponents, issue the following request:

```
_cmd: GET
_obj_class: log
_obj_id: logId S alarm
_scope: ALL
```

```
_filter: compId LEFT NI EM NODE1<space>
_filter: compId EQ NI EM NODE1
_attr_id: ALL
<blank line>
```

Note: The <space> represents a blank character and is required to filter out components such as EM NODE11, EM NODE12, EM NODE13 ... EM NODE19.

If the GET request is successful, a typical response message is:

```
_obj_class: alarmRecord
_obj_id: alarmRecordId I 28
_attr: time D 1994 01 31 17 54 47
_attr: compId NI EM NODE1 LP 7 X21 4
_attr: rawState E OOS
_attr: compCriticality I 40
_attr: alarmType E processing
_attr: faultCode S 70116501
_attr: severity E warning
_attr: probableCause E configurationError
_attr: notificationId I 117440514
_attr: originatorClass E passport
_attr: active I 1
_attr: customerId I 0
_attr: event E set
_block: _attr commentData S
Termination switch setting does not match provisioned.
Check switch setting or the provisioned data
(lineTerminationRequired).
_end_block:
_attr: fileName S PmsHwProcessHandler_Actor.cc
_attr: administrativeState E unlocked
_attr: operationalState E disabled
_attr: usageState E usageIdle
_attr: proceduralStatus E initializing
_attr: unknownStatus E unknownFalse
_attr: availabilityStatus E offline
_attr: standbyStatus E standbyEmpty
_attr: fileLineNumber I 451
_block: _attr fileVersion S
Oct 16 1993@ 00:28:47
_end_block:
```

```
_attr: processId S 7/0/22297
...
<blank line>
_end_resp: GET
_time: 1994 01 31 10 05 06
<blank line>
```

To retrieve all Passport alarms with a severity of *critical* and report only the *compId*, *severity*, and *faultCode* attributes, issue the following request:

```
_cmd: GET
_obj_class: log
_obj_id: logId S alarm
_scope: ALL
_filter: originatorClass EQ E passport
_filter: severity EQ E critical
_attr_id: compId
_attr_id: severity
_attr_id: faultCode
<blank line>
```

If the GET request is successful, a sample response is:

```
_obj_class: alarmRecord
_obj_id: alarmRecordId I 1884
_attr: compId NI EM NODER5 LP 11
_attr: faultCode S 70120200
_attr: severity E critical
<blank line>
_obj_class: alarmRecord
_obj_id: alarmRecordId I 1895
_attr: compId NI EM NODER5 SHELF $ CARD 11
_attr: faultCode S 70120100
_attr: severity E critical
...
<blank line>
_end_resp: GET
_time: 1994 01 31 10 05 06
<blank line>
```

The CREATE command

The CREATE command forms a single, new managed object. The CREATE request creates a sieve which reports changes for one of the following:

alarm
indicates a failure in the network or the correction of a previous failure

rawStateChange
indicates a changed value for a raw state attribute of a node managed object

ackStateChange
indicates a changed value for a ack state attribute of a node managed object

status
provides a periodic report of statistical data

> ***Note:*** The API user can use the GET, SET, CREATE, and DELETE services to control the reporting of events by manipulating sieve objects and their contents.

A CREATE request results in a single CREATE response, followed by an end-of-response message.

Creating a sieve to receive alarm event reports

To receive all alarm event reports, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S alarm
<blank line>
```

The following response message is returned. This acknowledges that the sieve creation was successful.

```
_obj_class: sieve
_obj_id: sieveId I 10
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
<blank line>
```

In this example, a sieveId of 10 has been created. Since the *admState* attribute is 1 by default, the sieve is unlocked, meaning it reports events as soon as they occur. Note that only one *eventType* may be specified for each sieve.

A sample alarm event report is:

```
_event_type: alarm
_sieve_id: 10
_obj_class: node
_obj_id: compId NI PM R70 PRI 26
_time: 1994 02 04 13 00 49
_attr: alarmType E operator
_attr: ncsDeviceType S RM
_attr: faultCode S 20304006
_attr: reporterName S CORENCS
_attr: ncsConditionMnem S OOS
_attr: severity E cleared
_attr: probableCause E operationalCondition
_attr: notificationId I 19485
_attr: originatorClass E dpn
_attr: active I 0
_attr: customerId I 0
_attr: event E clear
_attr: ncsNamsId I 4070
_attr: ncsSeverity E ncsMinor
_block: _attr commentData S
CO: R70_PI26
_end_block:
_block: _attr expertData S
EX: PROCESS ID:000E 0032 GLOBAL:00CF SEQUENCE:0023
_end_block:
_attr: ncsAction E ncsOperations
_attr: ncsSeqNum I 19485
_attr: correlatedNotifications I 19487
_attr: ncsDeviceName S R70
<blank Line>
```

Creating a sieve to receive alarm event reports with specific severity and faultCode

To receive all alarm event reports from the PM R73 node and its subcomponents, with a severity of *critical*, and a *faultCode* starting with 2030, issue the following request. This report includes only the *severity* and *faultCode* attributes.

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S alarm
_attr: eventFilter SS compId EQ NI PM R73
_attr: eventFilter SS compId LEFT NI PM R73
*


_attr: eventFilter SS faultCode LEFT S 2030
_attr: eventFilter SS severity EQ E critical
_attr: eventInfo S faultCode
_attr: eventInfo S severity
<blank line>
```

Note: The * character represents a blank character and is required to filter out components such as PM R74, PM R75, PM R76 ... PM R79.

If the CREATE request is successful, a sample response is:

```
_obj_class: sieve
_obj_id: sieveId I 10
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
```

A sample event report is:

```
_event_type: alarm
_sieve_id: 10
_obj_class: node
_obj_id: compId NI PM R73 PE 10
_time: 1994 02 04 13 00 49
_attr: faultCode S 20304006
_attr: severity E critical
<blank line>
```

Creating a sieve to receive link alarm event reports

To receive all link alarm event reports, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S alarm
_attr: repFilter SS objectClass EQ S link
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 10
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
<blank line>
```

A sample event report is:

```
_event_type: alarm
_sieve_id: 10
_obj_class: link
_obj_id: compId LI DBNL:PM AC2256 PE 1 PI 1 PO 4:PM
R220A-R22 PE 7 PI 7 PO 3:
_time: 1994 02 04 13 33 10
_attr: alarmType E equipment
_attr: ncsDeviceType S DBNL
_attr: faultCode S 10164021
_attr: reporterName S TOPNCS
_attr: ncsConditionMnem S ENABLED
_attr: severity E cleared
_attr: probableCause E equipmentFailure
_attr: notificationId I 767
_attr: originatorClass E dpn
_attr: active I 0
_attr: customerId I 0
_attr: event E clear
_attr: ncsNamsId I 4022
_attr: ncsSeverity E ncsWildcard
_block: _attr expertData S
EX: 0007 0107 005E 0025
_end_block:
_attr: ncsAction E ncsHardware
```

```
_attr: ncsSeqNum I 767
_attr: ncsDeviceName S AC2256/R22OA
<blank line
```

Creating a sieve to receive explicit cleared alarm notifications

To receive all alarm event reports including proxy clear alarms for all cleared alarms, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S alarm
_attr: explicitClears B true
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 10
<blank line>
_end_resp: CREATE
_time: 1999 02 03 18 50 02
<blank line>
```

Alarms are reported in a similar manner as the previous examples. In addition, reduced proxy clear alarms are issued to explicitly identify all cleared alarms and include the following information:

```
_event_type: alarm
_sieve_id: 5
_obj_class: node
_obj_id: compId NI PM A7104 PRI 11
_time: 1999 02 03 19 41 59
_attr: compCriticality I 50
_attr: faultCode S 20304003
_attr: severity E cleared
_attr: notificationId I 1859
_attr: originatorClass E clear
_attr: active I 0
_attr: customerId I 0
_attr: event E clear
_block: _attr commentData S
Explicited CLEAR Alarm generated by GMDR.
_end_block:
```

Creating a sieve to receive rawStateChange event reports

To receive all rawStateChange event reports, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S rawStateChange
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 2
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
<blank line>
```

A sample event report is:

```
_event_type: rawStateChange
_sieve_id: 2
_obj_class: node
_obj_id: compId NI PM R73 PE 8 PI 8 PO 3
_time: 1993 10 13 15 35 33
_attr: rawState E INSV
_attr: compCriticality I 40
<blank line>
```

Creating a sieve to receive ackStateChange event reports

To receive all ackStateChange event reports, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S ackStateChange
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 22
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
<blank line>
```

A sample event report is:

```
_event_type: ackStateChange
_sieve_id: 2
_obj_class: node
_obj_id: compId NI PM R72 PE 4 PI 4 PO 2
_time: 1993 10 13 15 35 33
_attr: compAckState E acked
<blank line>
```

Creating a sieve to receive ServerReset event reports

To receive all serverReset event reports, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S serverReset
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 44
_end_resp: CREATE
_time: 1996 03 28 17 49 39
<blank line>
```

A sample event report is:

```
_event_type: serverReset
_sieve_id: 44
_obj_class: server
_obj_id: serverId S GMDR
_time: 1996 03 28 17 50 06
_attr: resetType E dbReset
<blank line>
```

Creating a sieve to receive status event reports

To receive all status event reports from the direct subcomponents of the PM 1 PE 2 node, issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S status
_attr: repOClass S node
```

```
_attr: repOid SS compId NI PM 1 PE 2
_attr: repScope E next
<blank line>
```

If the CREATE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 2
<blank line>
_end_resp: CREATE
_time: 1994 02 04 12 52 25
<blank line>
```

A sample status event report is:

```
_event_type: status
_sieve_id: 2
_obj_class: node
_obj_id: compId NI PM 1 PE 2
_time: 1993 10 13 15 35 33
_attr: reporterName S CORENCS
_attr: active I 1
_attr: proType E pe386
_attr: proFreeQueue I 30
_attr: proFreeHeap I 60
_attr: proPPSTransmitted I 1000
_attr: proPPSReceived I 795
_attr: proCpuUtilization I 70
_attr: proServiceName S NCS386
_attr: statusType E processor
<blank line>
```

Getting the attributes of a sieve

To determine all the information about sieve 1, the API User must first create a sieve with attributes. Issue the following request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S alarm
_attr: repOClass S node
_attr: repOid SS compId NI EM NODE1
_attr: repScope S NEXT
_attr: annotation S sieve for critical alarms from\
```

EM NODE 1 next
_attr: eventInfo S severity
<blank line>

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
<blank line>
_end_resp: CREATE
_time: 1994 02 16 10 34 54
<blank line>
```

Now issue the following request:

```
_cmd: GET
_obj_class: sieve
_obj_id: sieveId I 1
_attr: ALL
<blank line>
```

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
_block: _attr annotation S
sieve for critical alarms from EM NODE1 next
_end_block:
_attr: admState I 1
_attr: eventFilter SS eventType EQ S alarm
_attr: eventInfo S severity
_attr: repOClass S node
_attr: repOid SS name NI PM NODE1
_attr: repScope S NEXT
<blank line>
_end_resp: GET
_time: 1994 02 09 12 40 31
<blank line>
```

Getting sieve annotation information

To retrieve the annotation information of sieve 1 (created in the previous example), issue the following request:

```
_cmd: GET
_obj_class: sieve
_obj_id: sieveId I 1
_attr: annotation
<blank line>
```

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
_block: _attr annotation S
sieve for critical alarms from EM NODE1 next
_end_block:
<blank line>
_end_resp: GET
_time: 1994 02 09 12 02 51
<blank line>
```

The SET command

The SET command modifies values of attributes of managed objects. A SET request results in zero or more SET responses (one for each selected object), followed by an end-of-response message.

The SET request changes the attributes of a sieve. Attributes control the type of information received; they have a name and an associated value. For a description of attributes, refer to the section on the managed object model in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Setting the sieve to locked

Locking a sieve means that the event reporting is stopped. The SET command can lock and unlock a sieve by changing the integer of the *admState* attribute to 0 (lock the sieve), or 1 (unlock the sieve).

In this example, the sieve is locked (meaning event reporting is stopped) by changing the *admState* attribute to 0 as follows:

```
_cmd: SET
_obj_class: sieve
_obj_id: sieveId I 3
_mod: REP admState I 0
<blank line>
```

If the SET request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 3
_attr: admState I 0
<blank line>
_end_resp: SET
_time: 1994 02 05 18 59 02
<blank line>
```

To restart the sieve, set the *admState* attribute to 1.

Setting the annotation attribute of a sieve

To change the annotation attribute of sieve number 5, issue the following request:

```
_cmd: SET
_obj_class: sieve
_obj_id: sieveId I 5
_mod: REP annotation S New Annotation String
<blank line>
```

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 5
_block: _attr annotation S New Annotation String
<blank line>
_end_resp: SET
_time: 1994 12 09 12 48 54
<blank line>
```

The DELETE command

The DELETE command deletes one or more managed objects. A DELETE request results in zero or more DELETE responses (one for each selected object), followed by an end-of-response message.

Deleting a sieve

To delete a sieve having an ID of 42, issue the following request:

```
_cmd: DELETE
_obj_class: sieve
_obj_id: sieveId I 42
<blank line>
```

If the DELETE request is successful, the typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 42
<blank line>
_end_resp: DELETE
_time: 1994 02 05 18 59 02
<blank line>
```

Note: A sieve can only be deleted by the same owner of the sieve and the sieve ID must match.

Appendix A
Compliance statement

This appendix indicates how the Alarm and Status API conforms to the general API functionality as described in 241-6001-200 *Preside MDM Application Programming Interface Primer*. It lists the various API Primer features, and indicates which features are supported. In addition, the statement describes the constraints, limitations, and conditions for the support of these features.

The tables in this appendix detail the following information:

- API message types
- API message lines
- API data types
- sieve object support

Each box in the Supported column of the tables is filled with one of the following letters:

- Y
  yes, the item is supported
- N
  no, the item is not supported
- C
  conditional, the item is supported under certain conditions. In this case, the condition is described.

API message types

The table “API message types” (page 88) indicates which messages are supported by the Alarm and Status API. This means that the message can be generated or received with all the mandatory lines. The Direction column indicates whether the message is sent or received by the API Provider.

Table 13
API message types

Message type	Direction	Supported
GET request	receive	C[1]
GET response	send	Y
SET request	receive	C[2]
SET response	send	Y
ACTION request	receive	Y
ACTION response	send	Y
CREATE request	receive	C[3]
CREATE response	send	Y
DELETE request	receive	C[4]
DELETE response	send	Y
REGISTER request	receive	Y
REGISTER response	send	Y
DEREGISTER request	receive	Y
DEREGISTER response	send	Y
EVENT-REPORT message	receive	C[5]
end-of-response message	send	Y
error message	send	Y
block message	send	Y
version message	send	Y
(Sheet 1 of 2)		

Table 13 (continued)
API message types

Message type	Direction	Supported
end message	send	Y
end message	receive	Y
(Sheet 2 of 2)		

- Scoping is not supported from the root object. The IN and NOTIN filter operators are not supported. Filtering is supported on all attributes. The filtering operations supported are based on the attribute type as follows:
  - I, D:
    P, EQ, NE, GT, LT, GE, LE
  - S, NI, LI:
    P, EQ, NE, LEFT, RIGHT, MIDDLE
  - B, E:
    P, EQ, NE
  - For statusRecord, only the EQ and NE operators are supported for the active attribute.
- The SET service is supported for the sieve object, for the admState, and the annotation attributes only. Filtering is not supported. The ADD, DEL, and DEF modification types are not supported.
- The CREATE service is supported for the sieve object. The object ID, superior object ID, and reference object ID CREATE attributes are not supported. They are ignored if they are provided in the CREATE request. Specify only one eventType eventFilter for each sieve. Alarm reporting sieves support the explicitClears Boolean parameter to force the generation of explicit alarm clearing notifications.
- The DELETE service is supported for the sieve object. Filtering is not supported.

- Only one type of event may be specified for each sieve. The repFilter attribute is supported only for the compId and the objectClass attributes. The repInfo sieve attribute, and the IN and NOTIN filter operators are not supported. Filtering is supported on all event attributes. The filtering operations supported are based on the attribute type as follows:

  I, D:
  P, EQ, NE, GT, LT, GE, LE

  S, NI, LI:
  P, EQ, NE, LEFT, RIGHT, MIDDLE

  B, E:
  P, EQ, NE

  For status notifications, only the EQ and NE operators are supported for the active attribute.

API message lines

The table “API message lines” (page 90) indicates which message lines and keywords are supported by the Alarm and Status API. It describes overall support for the message line.The table “Optional message lines” (page 94) indicates which optional message lines are supported by the Alarm and Status API for each API message type and for each API message line. Message lines that are mandatory are listed in this table, since there may be some conditions and restrictions when used.

Table 14
API message lines

Message line	Keywords	Supported
_action_type		Y
_attr		Y
_attr_id	general support ALL	Y Y
_block		Y
(Sheet 1 of 4)		

Table 14 (continued)
API message lines

Message line	Keywords	Supported
_capability_set		N
_cmd	REGISTER, DEREGISTER, ACTION, CREATE, SET, DELETE, GET	Y (see also the table “API message types” (page 88))
_echo		Y
_end	general support USER_REQUEST FATAL_ERROR SYSTEM_SHUTDOWN LOST_CONNECTION LOST_SESSION	Y Y Y Y Y Y
_end_block		Y
_error	general support ACCESS_DENIED APPLICATION_ERROR DUPLICATE_OBJECT INVALID_ACTION_TYPE INVALID_ATTRIBUTE_NAME INVALID_ATTRIBUTE_VALUE INVALID_FILTER INVALID_OBJECT_ID INVALID_OBJECT_CLASS INVALID_SCOPE LOGON_FAILS MISSING_ATTRIBUTE_VALUE MODIFICATION_ERROR NO_SUCH_OBJECT_ID OUT_OF_SEQUENCE	Y Y Y Y Y Y Y Y Y Y Y N Y Y Y Y
_event_type		Y

(Sheet 2 of 4)

Table 14 (continued)
API message lines

Message line	Keywords	Supported
_filter	general support P EQ NE LT GT LE GE LEFT MIDDLE RIGHT IN NOTIN	C[1] C[1] C[1] C[1] C[1] C[1] C[1] C[1] C[1] C[1] C[1] N N
_inv_id		Y
_mod	general support ADD DEL DEF REP	C[2] N N N Y
_obj_id		Y
_obj_class		Y
_password		N
_ref_obj_id		N
_scope	general support BASE NEXT ALL	Y Y Y Y
_sieve_id		Y
_sup_obj_id		N
_time		Y
_trace		Y
(Sheet 3 of 4)		

Table 14 (continued)
API message lines

Message line	Keywords	Supported
_user_id		Y
_version		Y
(Sheet 4 of 4)		

- Scoping is not supported from the root object. Filtering is supported on all attributes. Only one type of event may be specified for each sieve. The repFilter attribute is supported only for the compId and the objectClass attributes. The repInfo sieve attribute, and the IN and NOTIN filter operators are not supported. The filtering operations supported are based on the attribute type as follows:

  — I, D:
  P, EQ, NE, GT, LT, GE, LE

  — S, NI, LI:
  P, EQ, NE, LEFT, RIGHT, MIDDLE

  — B, E:
  P, EQ, NE

  — For statusRecord, only the EQ and NE operators are supported for the active attribute. For status notifications, only the EQ and NE operators are supported for the active attribute.

- The SET service is supported for the sieve object, for the admState, and the annotation attributes only. Filtering is not supported. The ADD, DEL, and DEF modification types are not supported.

Table 15
Optional message lines

Message type	Message line	Supported
GET request	_inv_id _scope _filter _attr_id	Y Y Y Y
GET response	_inv_id _attr	Y Y
SET request	_inv_id _scope _filter _mod	Y N N C[1]
SET response	_inv_id _attr	Y Y
ACTION request	_inv_id _scope _filter _attr	Y N N Y
ACTION response	_inv_id _attr	Y Y
CREATE request	_inv_id _obj_id _sup_obj_id _ref_obj_id _attr	Y N N N Y
CREATE response	_inv_id	Y
DELETE request	_inv_id _scope _filter	Y N N
DELETE response	_inv_id	Y
REGISTER request	_inv_id _user_id _password _attr	Y Y N N

(Sheet 1 of 2)

Table 15 (continued)
Optional message lines

Message type	Message line	Supported
REGISTER response	_inv_id _user_id _capability_set _attr	Y Y N N
DEREGISTER request	_inv_id	Y
DEREGISTER response	_inv_id _user_id _capability_set	Y Y N
EVENT-REPORT message	_attr	Y
end-of-response message	_inv_id	Y
error message	_inv_id _attr	Y Y
(Sheet 2 of 2)		

- The SET service is supported for the sieve object, for the *admState*, and the *annotation* attributes only. Filtering is not supported. The ADD, DEL, and DEF modification types are not supported. A SET request must include at least one *_mod*: line.

API data types

The table “API data types” (page 95) indicates the API data types supported by the Alarm and Status API. In this table, supported means that the base software to support the data type is present, whether or not there are actually any attributes of this type in the current object model.

Table 16
API data types

API data type	Supported
B (Boolean)	Y
I (Integer)	Y
(Sheet 1 of 2)	

Table 16
API data types

API data type	Supported
H (Hexadecimal)	N
D (Date/time)	Y
S (String)	Y
FS (Formatted String)	N
NI (Node Identifier)	Y
LI (Link Identifier)	Y
E (Enumerated	Y
SS (Sequence of Strings)	Y
SI (Sequence of Integers)	Y
SB (Sequence of Booleans)	N
RS (Range of Strings)	N
RI (Range of Integers)	N
(Sheet 2 of 2)	

Sieve object support

The table “Sieve object attributes” (page 96) indicates which attributes of the sieve object are supported by the Alarm and Status API.

Table 17
Sieve object attributes

Attribute	Supported
general support	Y
sieveId	Y
eventFilter	Y
admState	Y
annotation	Y
(Sheet 1 of 2)	

Table 17 (continued)
Sieve object attributes

Attribute	Supported
eventInfo	Y
repOClass	Y
repOid	Y
repScope	Y
repFilter	C[1]
repInfo	N
(Sheet 2 of 2)	

- Only one type of event may be specified for each sieve. The repFilter attribute is supported only for the compId and the objectClass attributes. The repInfo sieve attribute, and the IN and NOTIN filter operators are not supported. Filtering is supported on all event attributes. The filtering operations supported are based on the attribute type as follows:

  — I, D:
  P, EQ, NE, GT, LT, GE, LE

  — S, NI, LI:
  P, EQ, NE, LEFT, RIGHT, MIDDLE

  — B, E:
  P, EQ, NE

  — For status notifications, only the EQ and NE operators are supported for the active attribute.

Appendix B
Inbound Alarm API

This appendix describes the specialized capabilities of the Alarm Status API and GMDR to support the direct injection of external network management data for external devices. A mediation agent using these capabilities can be built to manage the external devices. External devices are neither DPN-100 nor Passport equipment.

There is another API, the IMDR API, which can also be used for injecting alarms and surveillance data from external devices and has a number of advantages over the Inbound Alarm API. If you are setting up alarm and surveillance injection for external devices, we recommend that you use the IMDR API instead. For information about the IMDR API, and the advantages it presents over the Inbound Alarm API, see “IMDR API” (page 113).

Inbound Alarm API

The architecture of the Inbound Alarm API allows:

- the injection of alarms and Raw State Change notifications into a General Management Data Router (GMDR) through the API interface
- the acknowledgment or unacknowledgment of alarms
- GMDR to be a provider for another GMDR. This allows the injection of an alarm using a hierarchical GMDR. For more information on GMDR, see the section on the General Management Data Router in 241-6001-310 *Preside MDM Server Reference Guide*.

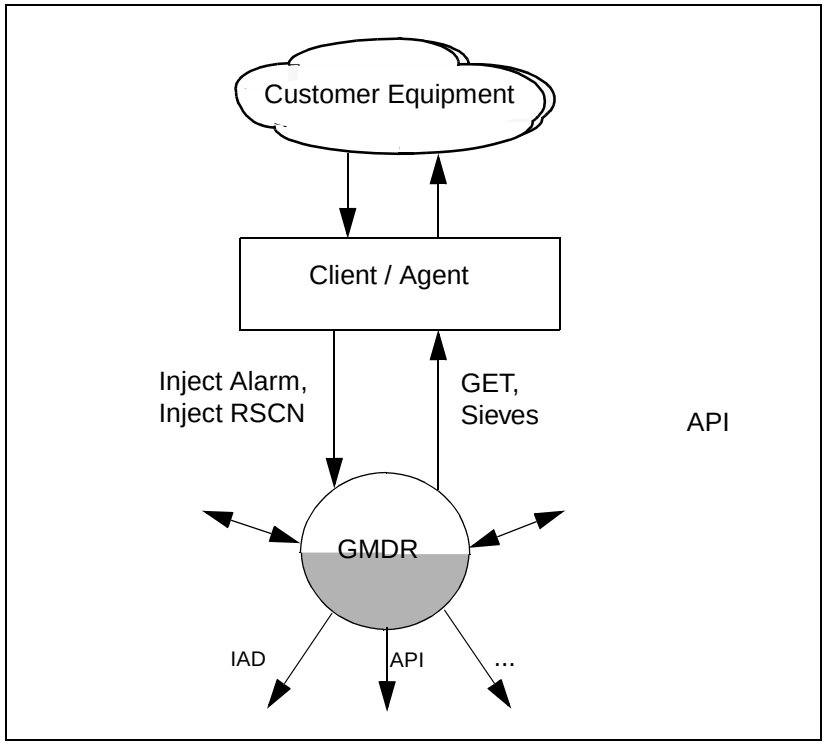
Preside Multiservice Data Manager (MDM) and customers can enhance their MDM surveillance capabilities by injecting non-DPN-100 and non-Passport management data into its data stream. This data is directly available to Alarm-

Based Surveillance. For State-Based Surveillance, the Network Model schema must be modified along with some internal mappings of the Network Viewer. For more information, refer to 241-6001-011 *Preside MDM Fault Management User Guide*. The use of the API interface for this purpose has the benefits of simplicity of use and that extracted and injected data use the same path and appear similar.

The Inbound Alarm API consists of two Action messages allowing you to inject Alarm and Raw State Change Notifications reporting. All other API and GMDR functions are available allowing GMDR to be used as specialized database for the External Device Management Mediator.

See the figure “Inbound Alarm API architecture” (page 100).

Figure 4
Inbound Alarm API architecture



GMDR External data interface

This section describes the new API ACTIONS: Alarm and Raw State Notification injection and alarm clearing.

Authentication

Before injecting an alarm, the External Device Management Agent or Client must first connect to the GMDR to which it is assigned, then register with the GMDR. To connect to the GMDR, invoke the API Provider by entering the following command line:

/opt/MagellanNMS/bin/gmdrapi [-h < hostname>] [-serv GMDR_<name>]

where:

`-h <hostname>`

should be used if the GMDR in question is found on another workstation.

`-serv GMDR_<name>`

should be used if the GMDR has an alternate service name.

The API Provider returns the following:

```
_version: 9.0 Alarm and Status API
```

To inject management data, the Agent must now register with GMDR by entering the *userCapability* attribute line in the REGISTER command:

Example

```
_cmd: REGISTER
_user_id: MODEMMgmntAgent
_attr: userCapability E mdInject
```

Note 1: The user name specified at REGISTER time allows you to identify the API Client in the GMDR Client dialog.

Note 2: If the userCapability line is not provided, you cannot inject alarms or state changes.

If the REGISTER request is successful, the API Provider returns the following:

```
_user_id: MODEMMgmntAgent
<blank line>
_end_resp: REGISTER
_time: 1994 11 29 16 52 29
```

Alarm and Raw State Notification injection

The addition of two new ACTIONs to the API protocol provides support for external data in GMDR. The first ACTION, *injectAlarm*, allows a client to inject an alarm into the Preside Multiservice Data Manager (MDM) surveillance stack. The second ACTION, *injectRawState*, allows a client to inject Raw State Notifications into the MDM surveillance stack. These two ACTIONs allow an external process to provide MDM with the information it needs to manage non-DPN-100 and non-Passport devices. It is the external process' responsibility to communicate with the managed devices and to convert their network management data, including the Alarm and component format mappings, into the proper API ACTIONs.

> ***Note:*** The specification of the component name is crucial as its value must uniquely identify the component in the network. The sequencing of category value pairs (all uppercase and blank separated) must also be strictly adhered to.

Since the API interface provides support for external data in GMDR, the external management process can use the GMDR database as if it were its own, enabling it to extract component states and active alarms if needed.

Injecting an Alarm Notification

After connecting and registering with GMDR, you can inject an alarm by issuing the following ACTION request:

```
[_invoke_id: <invoke ID>]
_cmd: ACTION
_obj_type: server
_obj_id: serverId S GMDR
_scope: BASE
_action_type: injectAlarm
_attr: originatorClass E <external>
_attr: compId NI <component ID>
```

```
_attr: event E <alarm event>
_attr: faultcode S <fault code>
_attr: severity E <alarm severity>
[_attr: notificationId I <alarm notification ID for
clears>]
[_attr: rawState E <component raw state>]
[_attr: alarmType E <alarm type>]
[_attr: time D <alarm data and time>
[_customerId I <CNMID>
[_attr: reporterName S <reporting agent name>
[_block: _attr commentData S
<comment string>
[_end_block:]
[_attr: clearScope E <alarmclearing scope>]
[_attr: correlatedNotifications I <cleared alarm's
notification ID]
[_attr: <other legal alarm attributes>]
```

Example

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0000001
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: rawState E OOS
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

Some missing attributes will be provided the following defaults:

- time - the current GMDR workstation time
- alarmType - *unknown*
- probableCause - *probCauseUnknown*
- active - 1 if event is set; otherwise, 0

- notificationId - unique value (for the life span of the server) deduced from an incrementing counter (value will be marked negative -- proxy - alarm behavior -- if the time was not specified)

If the *rawState* attribute is missing, GMDR will provide a rawState according the following rules:

- If the component has no active alarms, the rawState is assumed to be In Service (INSV).
- If all the active alarms of a component have *warning* or *minor* for severity, the rawState is assumed to be Troubled (TRB).
- If some of the active alarms of a component have *major* or *critical* for severity, the rawState is assumed to be Out Of Service (OOS).

If the ACTION request is successful, the API Provider will return an ACTION reply:

```
_obj_type: server
_obj_id: serverID S GMDR
<blank line>
_end_resp: ACTION
_time: 1994 11 20 16 52 29
```

If the request is unsuccessful, an error record is returned:

```
_error: APPLICATION_ERROR Could not inject alarm,
invalid or missing parameters
<blank line>
_end_resp: ACTION
_time: 1995 04 18 52 46
```

When injecting alarms, special care must be taken with the following attributes:

originatorClass - *external* alarms originate from an external device management agent. Alarms with external originatorClass are always displayed in the *Common* alarm format in the Alarm Display and the Component Information Viewer.

clearScope - for more information on the clearScope attribute, refer to “Clearing alarms with clearScope” (page 108).

Notification Id - must be a unique number at least within the reported module scope.

Faultcode - a string identifying the alarm code. It is recommended you use the DPN-NCS convention (also used by Passport) of eight hexadecimal uppercase digits. Codes in the A000 0000 to the AFFF FFFF range are reserved for external device management use.

Event - *set*, *messages*, and *clear* alarms are supported.

> ***Note:*** All of the alarm attributes follow the defined semantics and syntax of the object classes of the Alarm and Status API. For more information about object class definitions, refer to the table “AlarmRecord class attributes” (page 34).

When the Agent is connected to the GMDR with the API Provider, the Agent can issue all the ACTIONs, GETs, and sieve CREATEs it needs to perform its operations.

Injecting a Raw State Change Notification

After connecting and registering with GMDR, you can inject Raw State Change Notifications by issuing the following ACTION request:

```
[_invoke_id: <invoke ID>]
_cmd: ACTION
_obj_type: server
_obj_id: serverId s GMDR
_scope: BASE
_action_type: injectRawState
_attr: compID NI <component ID>
_attr: rawState E <component raw state>
```

Example

```
_cmd: ACTION
_obj_type: server
_obj_id: serverId s GMDR
_scope: BASE
_action_type: injectRawState
_attr: compID NI MDM MDMA53_8_2
_attr: rawState E INSV
```

In this example, the target component will be given a state of INSV unless it still has outstanding alarms.

Injecting a raw state of NEX on a module will remove all its alarms, the component, and its subcomponents from GMDR's database, which may be reflected in the network model. You may find it useful to inject a raw state of NEX to clear old components from the database upon reconnecting.

Clearing alarms

It is important when using external device management to clear alarms that have been resolved. The Inbound Alarm API allows you to clear resolved alarms in three ways:

- by injecting a *clear* alarm with correlatedNotifications of the corresponding *set* alarms
- by injecting a *clear* alarm with an appropriate *clearScope* ACTION attribute
- by using the Local Clear command provided by the *Alarm Display* and the *Component Information Viewer* tools. For more information on these tools, refer to the Alarm Display and Component Information Viewer sections in 241-6001-011 *Preside MDM Fault Management User Guide*.

Clearing alarms with correlatedNotifications

To clear known alarms you can inject new alarms with similar attributes, but with a *clear* value for the event attribute and the *correlatedNotifications* attribute which holds the *NotificationId* value of each *set* alarm to be cleared. Only correlatedNotifications whose value matches the NotificationId of an active *set* alarm of the same top-level component (module) is considered. This allows you to properly inform the operators of the reason for the clearing.

> ***Note 1:*** When using this method of clearing alarms, ensure the *event* attribute is given a value of *clear* and the *severity* attribute is set to *cleared*.
>
> ***Note 2:*** Alarm *notificationIds* and *correlatedNotifications* must be unique at least within the module or link for which they are generated.

Example

This is an example of an alarm that has been injected. It is followed by the message used to clear the alarm using correlatedNotificationIds.

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: event E set
_attr: faultcode S A0000003
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

To clear this alarm using the *correlatedNotifications* attribute, enter the following ACTION request:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0000003
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
_attr: correlatedNotifications I 123
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

This alarm is considered a clear due to the *clear* event and *cleared* severity attribute values. The *correlatedNotifications* attribute repeats the notificationId of the *set* alarm it is clearing.

Clearing alarms with clearScope

You can inject a clear alarm on a component by using the *clearScope* ACTION attribute. This clearScope attribute has four possible values: *clearBase*, *clearHier*, *clearNCSBase*, and *clearNCSHier*.

- *clearBase* - all alarms on the same component are automatically cleared
- *clearHier* - all alarms on the same component and all its subcomponents are automatically cleared
- *clearNCSBase* - all alarms on the same component matching the clear alarm's faultCode are automatically cleared. The NCS encoding scheme is assumed.
- *clearNCSHier* - all alarms on the same component and all its subcomponents matching the clear alarm's faultCode are automatically cleared. The NCS encoding scheme is assumed.

  Note: The *clearScope* is not an attribute of the alarm. It directs GMDR to automatically generate the proper correlatedNotifications for the cleared *set* alarms and adds them to the clear alarm.

The NCS Faultcode encoding scheme for *clear* alarms is as follows:

- Faultcodes consist of eight hexadecimal (uppercase) digits
- A clear alarm faultcode may contain two FF on even word boundaries. For instance, it can contain FF000000 and 00FF0000. These FF digits are considered wild cards and will match any faultcode digits in the same position. For example, FF000000 will clear 10000000 and 8F000000, but not 8F100000.
- The first four digits usually represent the area of the fault and the last four represent a specific fault within this area. Sending a clear alarm with a faultcode similar to A001FFFF will clear all alarms in area A001.

Example

This is an example of an alarm that has been injected.

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
```

```
_attr: compId NI MDM MDMA22_2_1
_attr: event E set
_attr: faultcode S A0000003
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

To clear this alarm using the *clearScope* attribute, enter either of the following ACTION requests:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A000FFFF
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
_attr: clearScope E clearNCSHier
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

Or:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s GMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0001000
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
```

```
_attr: clearScope E clearHier
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

Using Hierarchical GMDRs for injection

You can inject data into a top level GMDR, however, using a specifically configured GMDR server for injection has the following benefits:

- External alarms and components are segregated to their own database
- It is easier to distribute the external data by injecting it into a separate GMDR which is configured as a server to real GMDRs, the same as similarly configured DMDRs or FMDRs.
- Terminating the specific GMDR will clear all the corresponding alarms and component states from all upper GMDRs. The specific GMDR is therefore used as virtual private database for the external management data.

For more information on hierarchical GMDRs, see the section on the General Management Data Router in 241-6001-310 *Preside MDM Server Reference Guide*

Acknowledging and Unacknowledging Alarms

After connecting and registering with GMDR, you can acknowledge or unacknowledge alarms by issuing the following ACTION request:

```
[_invoke_id: <invoke ID>]
_cmd: ACTION
_obj_type: server
_obj_id: serverId s GMDR
_action_type: ackAlarm
_filter: compID <EQ/LEFT> NI <component ID>
[_filter: faultcode EQ S <fault code>]
[_filter: notificationId EQ I <alarm notification ID
for clears>]
_attr: ackFlag E <True|False>
[_attr: ackTime DT <TIME>]
[_attr: ackReason S <REASON>]
[_attr: ackUserId S <USER_ID>]
```

Example

```
_cmd: ACTION
_obj_type: server
_obj_id: serverId s GMDR
_action_type: ackAlarm
_filter: compID LEFT NI PM R72
_attr: ackFlag E True
_attr: ackReason S problem being investigated by Dan
_attr: ackUserId S dmei
```

If the ACTION request is successful, the API Provider will return an ACTION reply:

```
_obj_type: server
_obj_id: serverID S GMDR
_time: 1994 11 20 16 52 29

<blank line>
_end_resp: ACTION
_time: 1994 11 20 16 52 29
```

In this example, since there is a LEFT pattern match on the component ID, all active alarms for all components of the module, will become acknowledged. Also, as a result of this particular example will also cause appropriate and affected components to emit an *ackStateChange* notification to any client which has registered for the notification.

Note that the *ackReason* and *ackUserId* (and *ackTime*) are not mandatory and were not required in the above example.

To unacknowledge all of the alarms, it is only necessary to change the *ackFlag* value from True to False.

Appendix C
IMDR API

This appendix describes the specialized capabilities of the IMDR API to support the direct injection of surveillance data from external devices, which are not supported by Preside Multiservice Data Manager (MDM).

Advantages of the IMDR API over the Inbound Alarm API

This API presents a number of advantages over the Inbound Alarm API, described in “Inbound Alarm API” (page 99), which can also be used to inject alarm and surveillance information from external devices. These advantages include the following:

- Client applications running on the external devices can register with the API to clear alarms from the IMDR server if the connection to the injecting application is lost. This means that there can be no uncertain component states left behind to give a false reading of a component’s status.
- Like GMDR, the IMDR server handles injected data, but IMDR has the ability to handle actions that need to be forwarded to other servers.

Because of these advantages, we recommend that you use the IMDR API to support injection of surveillance data instead of the Inbound Alarm API when setting up injection from new external devices.

About the IMDR API

The IMDR server supports an applications interface (API) that provides a special set of services for client process to inject alarms and state-change notifications into IMDR. These special services are as follows:

- REGISTER requests for injection clients are equipped with a mandatory *userCapability* attribute with a value of *mdInject* to identify the client as an injection client. The request can also contain a *userDiscClean* attribute to indicate whether data injected by the client must be deleted should the client be disconnected.
- CREATE sieves are provided for local clear and property request notifications. These notifications replace the corresponding ACTION requests that IMDR would send to its subservers.
- ACTION requests are provided for injection. These include: inject alarm, inject raw state, and inject property.

Getting ready to use the API

To use the IMDR API for injecting surveillance data from an external device, perform the following steps:

- Configure and start the servers that support the IMDR API.

  See “Setting up servers to support the IMDR API” (page 115).

- Start the IMDR API.

  See “Starting the IMDR API” (page 116).

- Make the injecting client on the external device register with the API.

  See “Registering with the IMDR API” (page 117).

Once these steps are complete, the injecting client can send ACTION requests to inject surveillance information, as described in “Action requests” (page 118).

Setting up servers to support the IMDR API

Before an external device can inject surveillance information to the IMDR server through the IMDR's API, the application running on the external device must be set up to inject surveillance information, and the servers that support the API must be configured and started. Use the following procedure to configure and start these servers.

1. Set up the external device to inject surveillance data to a communications port on the workstation that runs the IMDR server.
2. Use the Server Manager Administration Tool to configure and start the IMDR server that supports the IMDR API. The format of the IMDR server's startup command is:

   `/opt/MagellanNMS/bin/imdr [-a] [-B] [-n <suffix>] \ [-b <buffer size>]`

   For an explanation of the options in the startup command, see the section on Injected Management Data Router (IMDR) in 241-6001-310 *Preside MDM Server Reference Guide*.

   For instructions to use the Server Manager Administration Tool, see the section on using the Server Administration tool in 241-6001-303 *Preside MDM Administrator Guide*.
3. Use the Server Manager Administration Tool to configure and start the PSERVER which is used to listen for data injected to a communications port on the workstation and route it to the API. This server must be configured to listen on the same communications port number specified when the external device was set up for injection. The format of the PSERVER's startup command is:

   `/opt/MagellanNMS/bin/pserver <port number> "<startup command>"`
4. Using the GMDR Administration Tool, configure GMDR to obtain surveillance information from IMDR. Use the following information:

   For Server Name, enter:
   IMDR (or IMDR_<suffix>)

   For Host Name, enter the IP address of the workstation that is running IMDR or its yellow pages name.

   For User/CapabilityID, enter:
   GMDR (or GMDR_<suffix>)

   Leave the Password field empty.

See the section on configuring GMDR to access the surveillance servers in 241-6001-303 *Preside MDM Administrator Guide*.

Starting the IMDR API

The command to start the IMDR is:

```
/opt/MagellanNMS/bin/imdrapi
[-f <filename>]
[-o | +o <filename>]
[-h <hostname>]
[-d]
[-k]
[-a]
[-w <width>]
[-echo]
[-v}
[-help]
```

where:

`[-f <filename>]`

specifies an input file that contains a set of requests for the API to execute

`[-o <filename>| +o <filename>]`

specifies an output file into which the API writes input and output information. You can specify two types of writing: -o overwrites existing information in the file, and +o appends information to the end of existing information in the file.

`[-h <hostname>]`

specifies the hostname of the workstation that runs the API server

`[-d]`

runs the API in daemon mode. In daemon mode, the API does not exit at the end of an input operation

`[-k]`

runs the API in keyboard mode. In keyboard mode, the API reads from standard input when it reaches the end of an input file

`[-a]`

runs the API in asynchronous mode. In asynchronous mode, the API does not wait for a query to complete before it executes the next request.

`[-w <width>]`

specifies the width of lines displayed on the screen or output to a file in characters (1 to 255)

`[-echo]`

echoes input and output information to the screen

`[-v]`

enables verbose mode and is used for interactive sessions

`[-help]`

outputs help information

Registering with the IMDR API

To inject surveillance data, the client application running on the external device must register with IMDR by supplying the API with a REGISTER command containing a *userCapability* attribute line with a value of *mdIject* that identifies the client as an injecting client. An additional line containing a *userDiscClean* attribute can be included to indicate whether data injected by the client is to be cleared from IMDR if the connection to the client is lost.

A register request takes the following form:

```
_cmd: REGISTER
_user_id: <application user id>
_attr: userCapability E mdInject
_attr: userDiscClean B <True |False>
```

Example

```
_cmd: REGISTER
_user_id: ABXYZ
_attr: userCapability E mdInject
_attr: userDiscClean B True
```

Note: If the userCapability line is not provided, you cannot inject alarms or state changes.

If the REGISTER request is successful, the API returns a response similar to the following:

```
_user_id: ABXYZ
<blank line>
_end_resp: REGISTER
_time: 1994 11 29 16 52 29
```

Action requests

Once the injecting client has connected and registered with IMDR through the API, the client can send action requests to IMDR to inject alarm, raw state change, and component property notifications.

Injecting an alarm notification

After connecting and registering with IMDR, a client can inject an alarm by issuing an ACTION request in the following form:

```
[_invoke_id: <invoke ID>]
_cmd: ACTION
_obj_type: server
_obj_id: serverId S IMDR
_scope: BASE
_action_type: injectAlarm
_attr: originatorClass E SNMP
_attr: compId NI <component ID>
_attr: event E <alarm event>
_attr: faultcode S <fault code>
_attr: severity E <alarm severity>
_attr: notificationId I <alarm notification ID for
clears>
[_attr: rawState E <component raw state>]
[_attr: alarmType E <alarm type>]
[_attr: time D <alarm date and time>
[_customerId I <CNMID>
```

```
[_attr: reporterName S <reporting agent name>
[_block: _attr commentData S
<comment string>
[_end_block:]
[_attr: clearScope E <alarmclearing scope>]
[_attr: correlatedNotifications I <cleared alarm's
notification ID]
[_attr: <other legal alarm attributes>]
```

Example

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0000001
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: rawState E OOS
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

Some missing attributes will be provided the following defaults:

- time - the current IMDR workstation time
- alarmType - unknown
- probableCause - probCauseUnknown
- active - 1 if event is set; otherwise, 0

If the rawState attribute is missing, IMDR will provide a rawState according the following rules:

If the component has no active alarms, the rawState is assumed to be In Service (INSV).

If all the active alarms of a component have warning or minor for severity, the rawState is assumed to be Troubled (TRB).

If some of the active alarms of a component have major or critical for severity, the rawState is assumed to be Out Of Service (OOS).

If the ACTION request is successful, the API returns an ACTION reply, similar to the following:

```
_obj_type: server
_obj_id: serverID S IMDR
<blank line>
_end_resp: ACTION
_time: 1994 11 20 16 52 29
```

If the request is unsuccessful, an error record is returned similar to the following:

```
_error: APPLICATION_ERROR Could not inject alarm,
invalid or missing parameters
<blank line>
_end_resp: ACTION
_time: 1995 04 18 52 46
```

When injecting alarms, special care must be taken with the following attributes:

- originatorClass - external alarms originate from an external device management agent. Alarms with external originatorClass are always displayed in the Common alarm format in the Alarm Display and the Component Information Viewer.
- clearScope - for more information on the clearScope attribute, refer to “Clearing alarms with clearScope” on page 149.
- Notification Id - must be a positive integer and unique at least within the reported module scope.
- Faultcode - a string identifying the alarm code. Codes in the A000 0000 to the AFFF FFFF range are reserved for external device management use.

- Event - set, messages, and clear alarms are supported.

  All of the alarm attributes follow the defined semantics and syntax of the object classes of the Alarm and Status API.

  When the Agent is connected to the IMDR with the API, the Agent can issue all the ACTIONs and sieve CREATEs it needs to perform its operations.

Injecting a Raw State Change Notification

After connecting and registering with IMDR, a client application can inject Raw State Change Notifications by issuing an ACTION request in the following form:

```
[_invoke_id: <invoke ID>]
_cmd: ACTION
_obj_type: server
_obj_id: serverId S IMDR
_scope: BASE
_action_type: injectRawState
_attr: compID NI <component ID>
_attr: rawState E <component raw state>
```

Example

```
_cmd: ACTION
_obj_type: server
_obj_id: serverId s IMDR
_scope: BASE
_action_type: injectRawState
_attr: compID NI MDM MDMA53_8_2
_attr: rawState E INSV
```

Injecting a raw state of NEX on a module will remove all its alarms, the component, and its subcomponents from IMDR's database, which may be reflected in the network model. You may find it useful to inject a raw state of NEX to clear old components from the database upon reconnecting.

Clearing alarms

It is important when using external device management to clear alarms that have been resolved. IMDR API lets an injecting client clear resolved alarms in three ways:

- by injecting a *clear* alarm with *correlatedNotifications* of the corresponding *set* alarms
- by injecting a *clear* alarm with an appropriate *clearScope* ACTION attribute
- by injecting a clear alarm with the same fault code as the SET alarm that set the alarm in the first place

Clearing alarms with correlatedNotifications

To clear known alarms a client can inject new alarms with similar attributes, but with a *clear* value for the event attribute and the *correlatedNotifications* attribute which holds the *NotificationId* value of each *set* alarm to be cleared. Only *correlatedNotifications* whose value matches the *NotificationId* of an active *set* alarm of the same top-level component (module) is considered. This lets the client properly inform the operators of the reason for the clearing.

Note 1: When using this method of clearing alarms, ensure the *event* attribute is given a value of *clear* and the *severity* attribute is set to *cleared*.

Note 2: Alarm *notificationIds* and *correlatedNotifications* must be unique at least within the module or link for which they are generated.

Example

This is an example of an alarm that has been injected. It is followed by the message used to clear the alarm using correlatedNotificationIds.

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: event E set
_attr: faultcode S A0000003
```

```
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

To clear this alarm using the *correlatedNotifications* attribute, enter the following ACTION request:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0000003
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
_attr: correlatedNotifications I 123
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

This alarm is considered a clear due to the *clear* event and *cleared* severity attribute values. The *correlatedNotifications* attribute repeats the *notificationId* of the *set* alarm it is clearing.

Clearing alarms with clearScope

A client can inject a clear alarm on a component by using the *clearScope* ACTION attribute. This *clearScope* attribute has four possible values: *clearBase*, *clearHier*, *clearNCSBase*, and *clearNCSHier*.

- *clearBase* - all alarms on the same component are automatically cleared
- *clearHier* - all alarms on the same component and all its subcomponents are automatically cleared

- *clearNCSBase* - all alarms on the same component matching the clear alarm's faultCode are automatically cleared. The NCS encoding scheme is assumed.
- *clearNCSHier* - all alarms on the same component and all its subcomponents matching the clear alarm's *faultCode* are automatically cleared. The NCS encoding scheme is assumed.

Note: The *clearScope* is not an attribute of the alarm. It directs IMDR to automatically generate the proper *correlatedNotifications* for the cleared *set* alarms and adds them to the clear alarm.

The *NCS faultcode* encoding scheme for *clear* alarms is as follows:

- Faultcodes consist of eight hexadecimal (uppercase) digits
- A clear alarm faultcode may contain two FF on even word boundaries. For instance, it can contain FF000000 and 00FF0000. These FF digits are considered wild cards and will match any faultcode digits in the same position. For example, FF000000 will clear 10000000 and 8F000000, but not 8F100000.
- The first four digits usually represent the area of the fault and the last four represent a specific fault within this area. Sending a clear alarm with a faultcode similar to A001FFFF will clear all alarms in area A001.

Example

This is an example of an alarm that has been injected.

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: event E set
_attr: faultcode S A0000003
_attr: severity E major
_attr: notificationId I 123
_attr: event E set
_attr: reporterName S MDEMMgmntAgent
_attr: alarmType E equipment
```

```
_block: _attr commentData S
The MODEM has lost power.
_end_block:
```

To clear this alarm using the *clearScope* attribute, enter either of the following ACTION requests:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A000FFFF
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
_attr: clearScope E clearNCSHier
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

Or:

```
_cmd: ACTION
_action_type: injectAlarm
_obj_type: server
_obj_id: serverId s IMDR
_attr: originatorClass E external
_attr: compId NI MDM MDMA22_2_1
_attr: faultcode S A0001000
_attr: severity E cleared
_attr: notificationId I 124
_attr: event E clear
_attr: reporterName S MODEMMgmntAgent
_attr: clearScope E clearHier
_block: _attr commentData S
The MODEM is back in service.
_end_block:
```

Index

R

S

T

U

X

Preside Multiservice Data Manager

Alarm and Status API

Reference Guide

R14.3

Publication: 241-6001-203
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003
Printed in Canada

NORTEL NETWORKS