NORTEL NETWORKS

Preside Multiservice Data Manager

Network Model API

Reference Guide

241-6001-201

Preside Multiservice Data Manager

Network Model API

Reference Guide

Publication: 241-6001-201
Document status: Standard
Document version: 15.1RSUP
Document date: August 2004

Printed in Canada

Publication history

August 2004

15.1 RSUP Standard
Commercial availability except for MPE support which will be available in a future release.

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 Network Model Application Programming Interface (API). This document describes the Network Model 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

241-6001-201 *Preside MDM Network Model API Reference Guide* contains the following sections:

- "Introducing the Network Model API" (page 13) describes the managed objects, API messages, and the event model of the Network Model API.
- "Object model" (page 21) describes what objects are managed in the Network Model API.
- "Using the Network Model API" (page 59) describes how you use the Network Model API.
- "Compliance statement" (page 85) details how the Network Model API complies with the functionality described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.
- "Error messages" (page 95) provides a list of error messages for the Network Model API.

What's new in this document

There have not been any new changes to this NTP 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:

- 241-6001-015 *Preside MDM Network Model Administrator Guide*
- 241-6001-100 *Preside MDM Installation*
- 241-6001-200 *Preside MDM Application Programming Interface Primer*

Chapter 1
Introducing the Network Model API

This section provides an introduction to the Network Model API. This section contains the following information:

What is the Network Model API

The Network Model API (Application Programming Interface) provides a well-defined and stable application programming interface for communicating with the Preside Multiservice Data Manager (MDM) Network Model, so that you can easily write custom applications.

The Network Model API is an ASCII interface that provides access to the following:

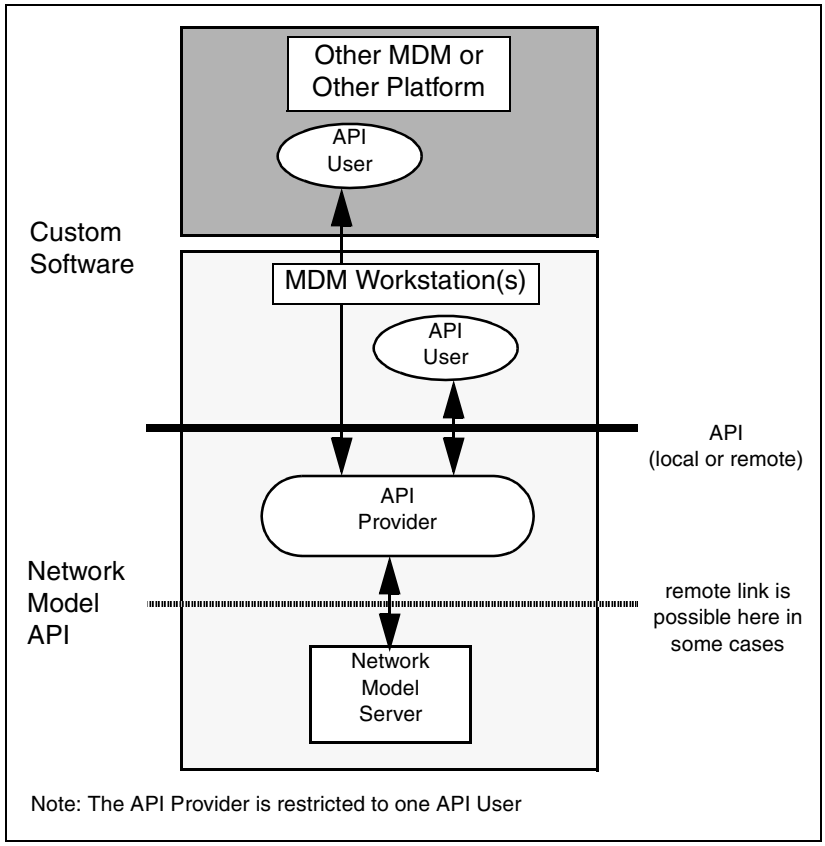
- state of each Network Model component
- topology of the Network Model
- attribute value information for each Network Model component
- possible types of Network Model components
- possible attributes (and their values) for each of the Network Model component types
- hierarchical structure of the Network Model component types
- notification of network and raw state changes for each Network Model component

- notification of network-wide changes such as Network Model component creation and deletion

At least two programs are typically involved in the Network Model API customer application. One is the Network Model API, and the other is the custom developed application. The Network Model API is message-based, and can be remotely accessed for custom programming from either an Preside Multiservice Data Manager (MDM) workstation platform or an ASCII-based workstation interface platform.

The Network Model API is one of many MDM APIs. The MDM workstation custom programming environment is shown in the figure "DPN-MDM workstation custom programming environment" (page 15). The common behavior and rules for all MDM APIs are described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 1
DPN-MDM workstation custom programming environment



The Network Model data is accessed through the Network Model Server, which finds the data and delivers it to the Network Model API.

The API Provider is typically called from an operating system shell to query data, matching a given set of conditions, from the Network Model. These conditions are specified either on the command line itself, or in an input file using an API syntax. In the latter case, many queries can be executed in a single call to the Network Model API Provider.

The input API may also be read from standard input (*stdin*) allowing the API Provider to be inserted in a set of piped commands in a UNIX shell since its output may also be in API format on the standard output (*stdout*) stream. Alternatively, the output may be directed to files. The customer application must invoke the API Provider, either through the pipes, or by invoking the command line options. The customer application requests the API Provider's services by sending its commands in the API format. The customer application that parses output from an API Provider must meet the rules described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

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.

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. Also, the API Provider sends a version message during initialization, and an end message during termination. The API sends an end message to request termination.

Object classes

The Network Model API, has the following classes of objects:

- node
- link
- org
- orgNode
- orgLink
- sieve
- model
- nodeType
- linkType

- attrType

For a description of the managed objects, see “Object classes” (page 21).

Action types

The Network Model API has no action types.

Sieves and event reports

The sieve objects of the Network Model API control the flow of notifications to the API user.

The event report types of the Network Model API are:

- **ackLevelChange**
  consist of acknowledgement level (acknowledged or maintenence mode) changes for the components and links
- **networkChange**
  consist of the network-wide changes such as Preside Multiservice Data Manager (MDM) Network Model component creation and deletion
- **networkStateChange**
  consist of network (propagated) state changes for each MDM Network Model component
- **rawStateChange**
  consist of raw state changes for each MDM Network Model component

Message types

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.

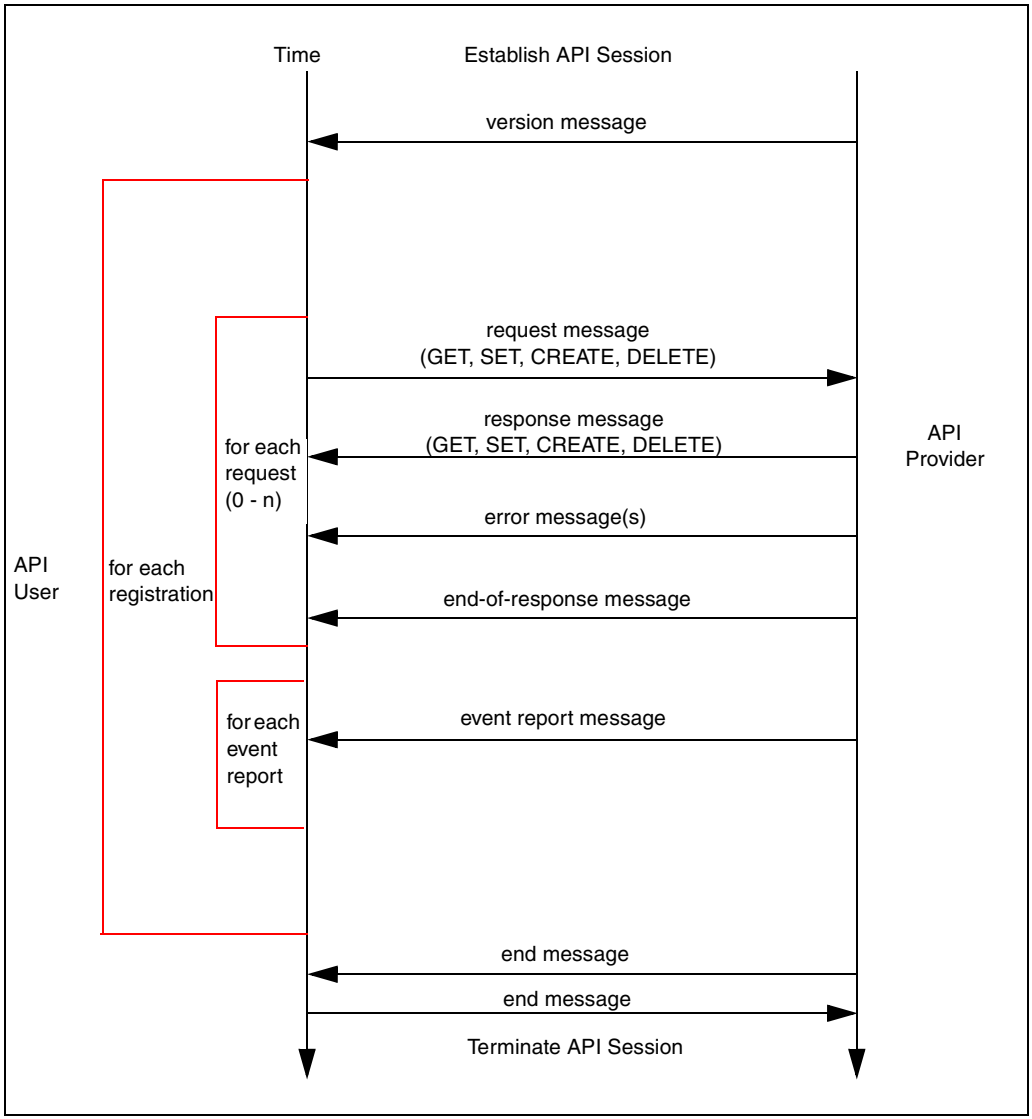
The end-of-response message indicates that no more responses are forthcoming from the API Provider and that the request processing is complete. See the figure "Time line diagram" (page 19) for an overview of the session events.

For the Network Model API, the message types are:

- **CREATE**
  The CREATE message is only supported for the sieve object. The *obj_id*:, *sup_obj_id*:, and *ref_obj_id*: message lines are not supported.
- **DELETE**
  The DELETE message is only supported for the sieve object. Scoping and filtering are not supported.
- **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 request.
  The filtering restrictions are class dependant and are documented in the tables in "Object model" (page 21).
  Access to the sieve information is restricted to the sieve creators.
- **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**
  More than one type of event report may be specified for each sieve. If no event type is specified, then the resulting sieve reports all the possible event types.
  The repFilter is subject to the same restrictions as the restrictions documented for the GET filtering in "Object model" (page 21).

For more details on the message types, see 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 Network Model API object model. This section contains the following information:

Object classes

The Network Model API provides topology, database information, state change notifications, and some configuration information to API Users through the following classes of objects:

network The *network* object represents the containers of the physical and organizational views of the Network Model. See also *241-6001-200 Preside MDM Application Programming Interface Primer*.

typeSet The *typeSet* object represents the containers of the linkType, nodeType, and attrType objects. See also 241-6001-200 *Preside MDM Application Programming Interface Primer*.

model The *model* object represents the Network Model as a database. It contains information about the state of the database, the model name and file path. The model object emits notifications about global changes of the network model.

org The *org* object represents the containers of the organizational hierarchies.

orgNode The *orgNode* object represents organizational level components like the region and site objects contained within the Network Model system. These objects emit state change notifications.

orgLink The *orgLink* object represents links between organizational level components. The region and site links contained within the Network Model system are examples of such objects. These objects emit state change notifications.

node The *node* object represents Preside MDM managed components (modules, services, and their subcomponents). These objects emit state change notifications, provide status and some general configuration information such as, IMAGE, PROCESSOR_TYPE, and/or SHELF_TYPE, as appropriate.

link The *link* object represents trunks and links between the Preside Multiservice Data Manager (MDM) managed components. These objects emit state change notification, provide status and endpoint information, as appropriate.

nodeType The *nodeType* object provides information concerning the possible attributes of the node class instances.

linkType The *linkType* object provides information concerning the possible attributes of the link class instances.

attrType The *attrType* object provides information concerning the possible values of the info attributes for the node and link class instances.

sieve The *sieve* object controls the flow of notifications to the API Users. For details, see the section on sieve object class definitions in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

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 containment hierarchy consists of the following parts:

- network containment hierarchy
- nodeType containment hierarchy
- linkType containment hierarchy
- attrType containment hierarchy
- sieve containment hierarchy
- model containment hierarchy

Network containment hierarchy

The Network containment hierarchy has two views; physical and organizational. The figure "Structure of the Network Model containment hierarchy" (page 25) shows these two views and the naming tree.

Physical view

The physical view starts under the network node named *compRoot* and contains the core of the Network Model. The core consists of modules, hardware and software subcomponents, and the links between them. The node object class represents the modules, the hardware and software subcomponents. The link object class represents the links between the node objects.

Each module is represented by a node object located directly under the compRoot object. Module nodes may contain hardware and/or software subcomponent nodes in a hierarchical structure (example, PM-PE-PI-PO). Subcomponent nodes may contain other subcomponent nodes. Each link is represented by a link object located directly under the compRoot object.

Figure 3
Structure of the Network Model containment hierarchy

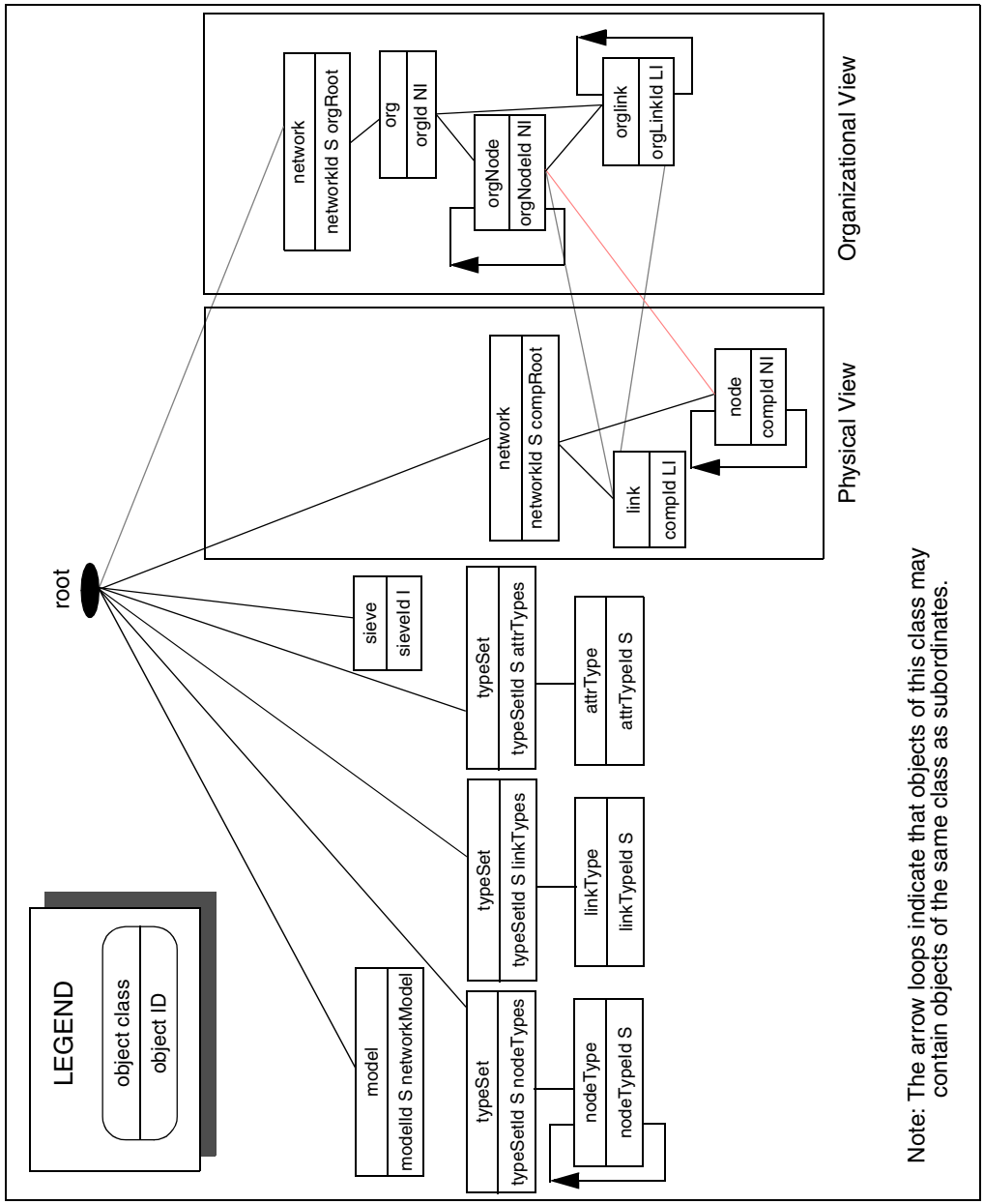
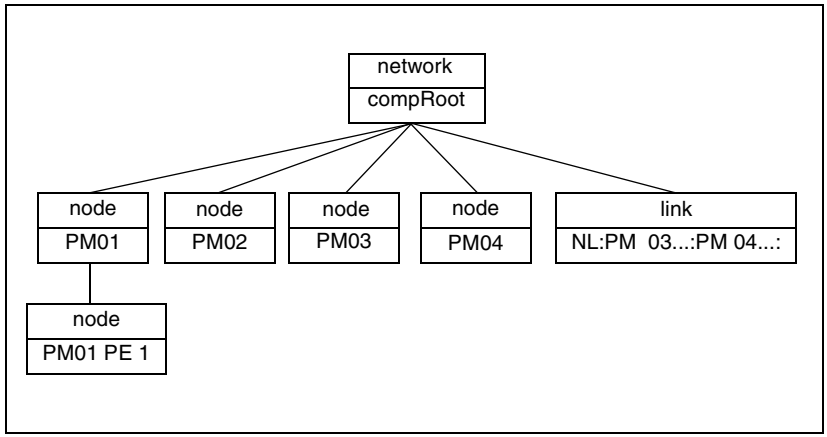


Figure 4
Example of a physical view



Organizational view

The organizational view provides different views of the physical network by modelling the physical resources as part of different hierarchies.

An organization provides a view of some part of the network. Any particular real network resource, a Packet Module (PM) and its subcomponents for example, can be contained within an arbitrary number of different organizations.

For each organization, there is a hierarchy of organizational level objects (*orgNode* and *orgLink* objects) located under an organization object (*org* object).

Example

In a DPN organization, there are SITE organizational nodes contained within the REGION organizational node objects, contained in an organization object. The organization objects are contained within the orgRoot network object. See the figure “Example of Network Model organizational view” (page 28).

The org object class is a container for an organizational hierarchy. The organization level components of an organizational view (below the organization and above the module level) are modeled as the orgNode class objects.

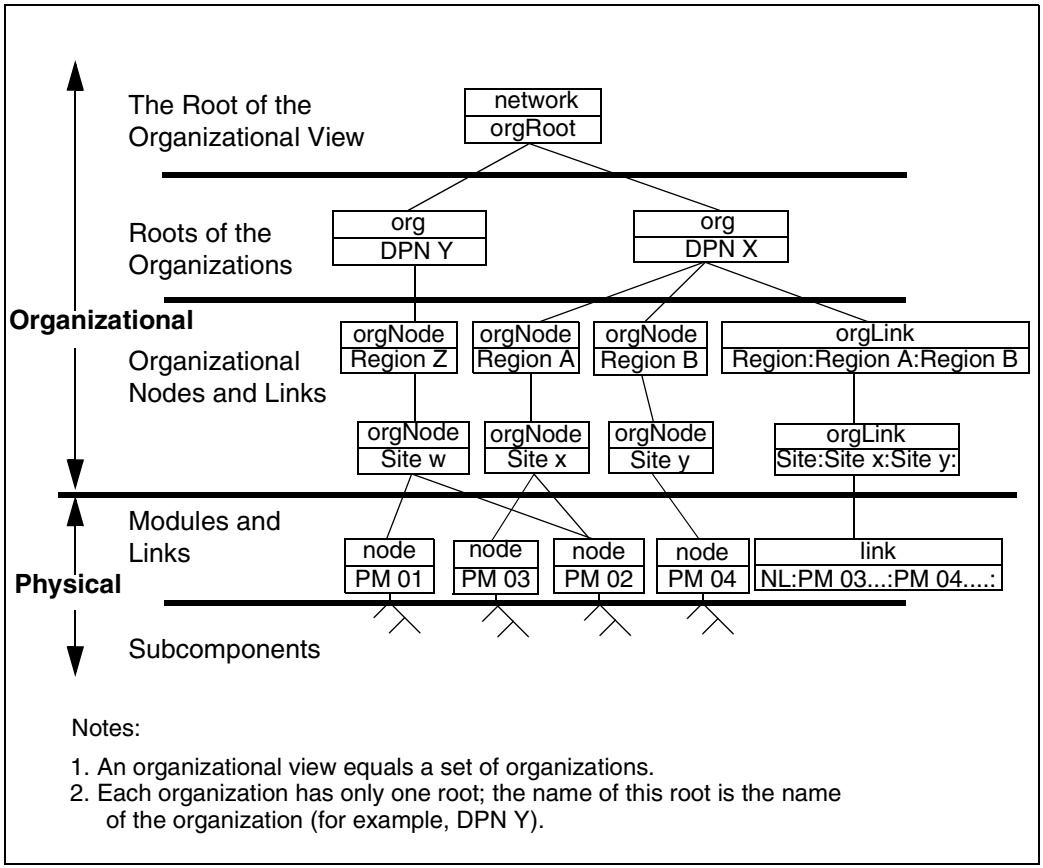
The links between the organizational nodes are modeled as instances of the orgLink object class. An organizational link is a virtual link; it represents the set of links between the next lower level of nodes in the organizational hierarchy.

Example

A region link represents all the site links between the sites in one region and the sites in another region.

An organization may be partitioned into regions, and then further partitioned into sites. Sites contain network resources such as Packet Modules (PMs) and Operations Agents (OAs). Within a given organization, a particular network resource may belong to one site, but that particular network resource may belong to multiple organizations. A site belongs to only one organization.

Figure 5
Example of Network Model organizational view



Notes:

1. An organizational view equals a set of organizations.
2. Each organization has only one root; the name of this root is the name of the organization (for example, DPN Y).

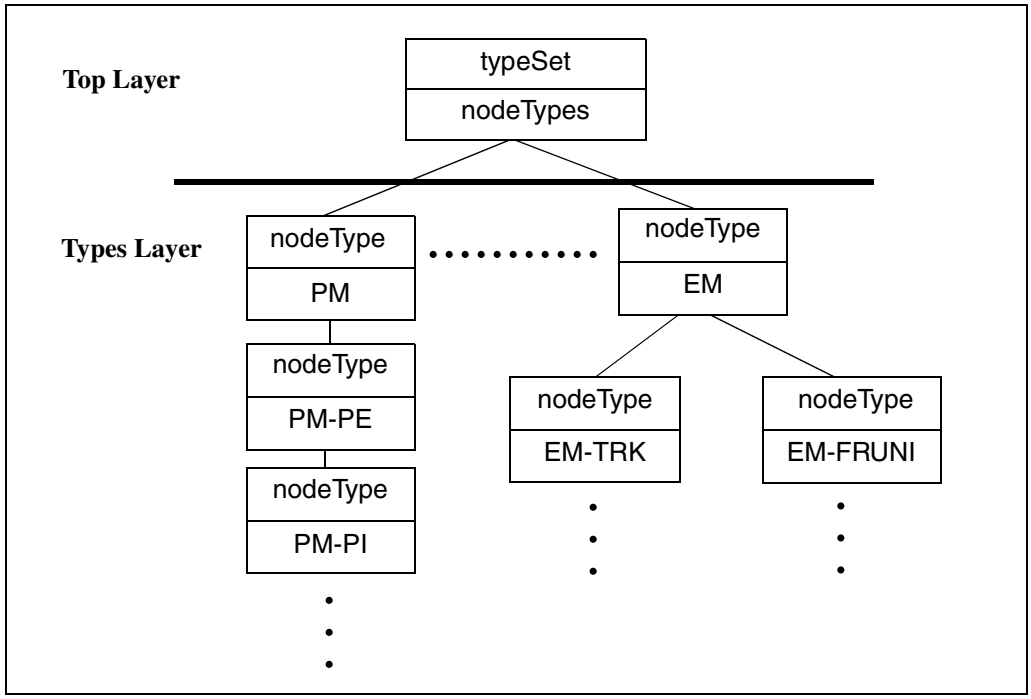
NodeType containment hierarchy

The *nodeType* containment hierarchy is a hierarchy of objects of nodeType class only.

The *nodeType* containment hierarchy consists of two layers: the top layer and the types layer (see the figure "NodeType containment hierarchy" (page 29)). The top layer represents the root of the nodeType containment hierarchy. This unique object is modeled as a type of typeSet, is named by nodeTypes, and is only used as a scoping base to get access to the lower level nodeType objects.

Each instance in the types layer represents one type of node in the Preside Multiservice Data Manager (MDM) Network Model. These objects are modeled as nodeType class objects.

Figure 6
NodeType containment hierarchy

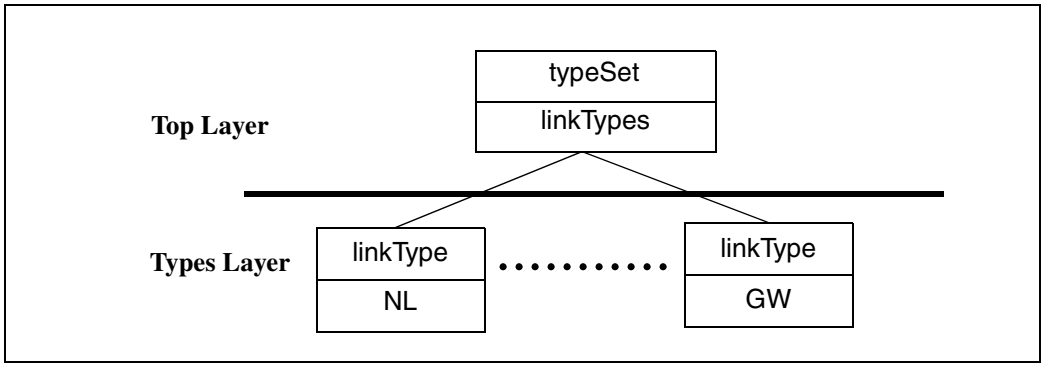


LinkType containment hierarchy

The *linkType* containment hierarchy consists of two layers: the top layer and the types layer (see the figure "LinkType containment hierarchy" (page 30)). The top layer represents the root of the linkType containment hierarchy. This unique object is modeled as a class of *typeSet*, is named by linkTypes, and is only used as a scoping base to get access to the lower level linkType objects.

Each instance in the types layer represents one type of link in the Preside Multiservice Data Manager (MDM) Network Model. These objects are modeled as linkType class objects.

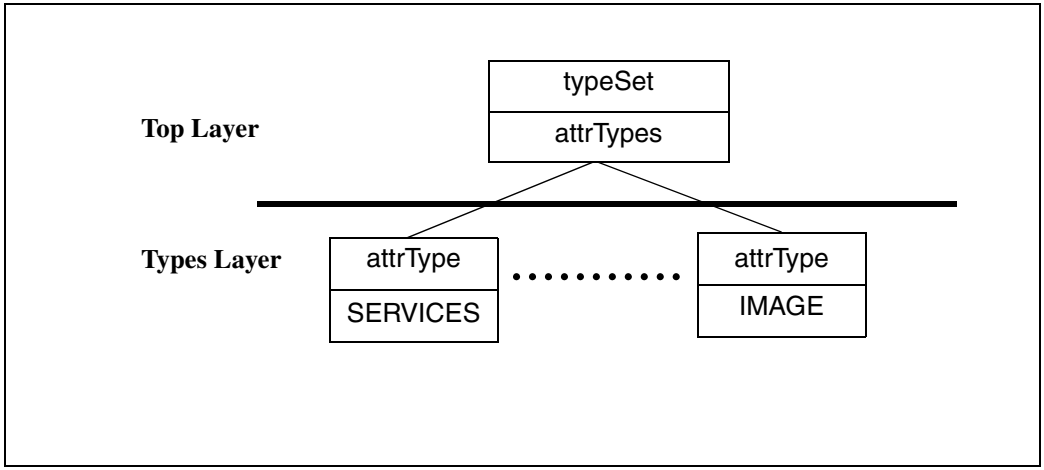
Figure 7
LinkType containment hierarchy



AttrType containment hierarchy

The attrType containment hierarchy consists of two layers: the top layer and the types layer (see the figure “AttrType containment hierarchy” (page 30)). The top layer represents the root of attrType containment hierarchy. This unique object is modeled as an object type of typeSet, is named by attrTypes, and is only used as a scoping base to get access to the lower level attrType objects. Each instance in the types layer represents one info attribute of the Network Model. These objects are modeled as attrType class objects.

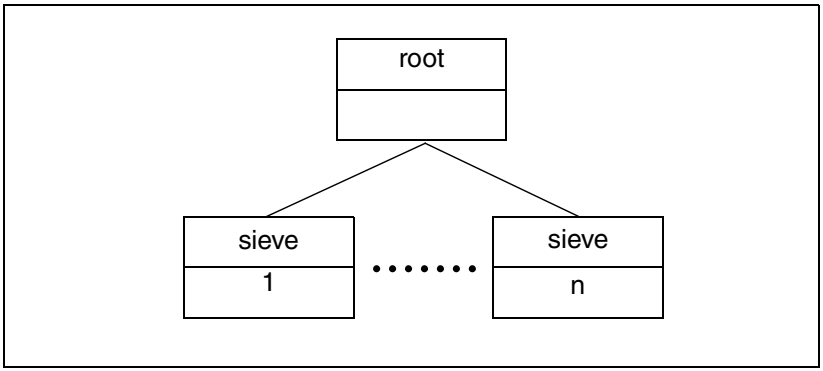
Figure 8
AttrType containment hierarchy



Sieve containment hierarchy

The *sieve* objects control the flow of notifications to the API User. The sieve containment hierarchy is a flat hierarchy with all objects of type sieve under the root object (see the figure “Sieve containment hierarchy” (page 31)). Since only base scoping is allowed (see “Compliance statement” (page 85)), sieve objects can only be addressed using their names.

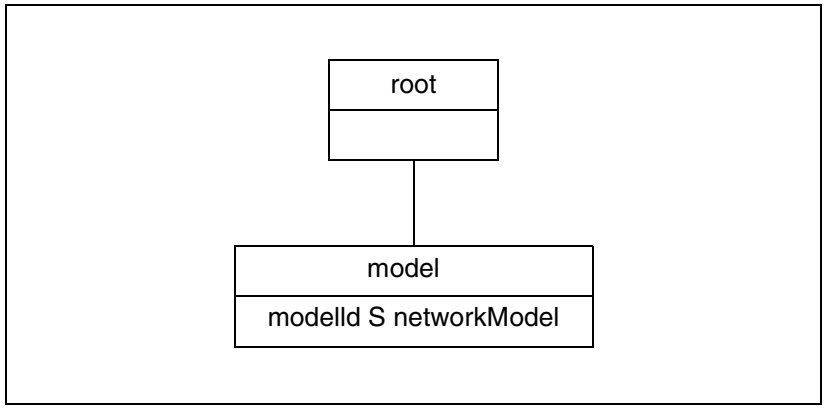
Figure 9
Sieve containment hierarchy



Model containment hierarchy

The *model* object represents the model database as an object. The model containment hierarchy contains a unique object located under the root object (see the figure “Model containment hierarchy” (page 32)). Since scoping from the root is not permitted (see “Compliance statement” (page 85)), the model object can only be addressed using its name.

Figure 10
Model containment hierarchy



Object class definitions

The following tables present the attributes of each object class as well as the notifications that can be emitted by these objects.

The table "Attribute list and notification list table format" (page 34) shows the attribute list and the notification list.

The first half of the table (the core and info attribute list) contains four columns. The first column is the name of the attribute, the second column is the data type of the attribute value, the third column indicates if filtering is allowed on that attribute and what operators can be used, and the fourth column gives an explanation of the attribute significance.

The filter column indicates that attribute filtering is supported on a GET request as well as by the sieve reporting object filtering capability. For example, the API User requires all the nodes and links with a CUSTOMER_ID equal to 1. To do this, the API User can specify an EQ filter on the CUSTOMER_ID attribute of the node object class while making a GET request. An example of this request follows:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
```

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

Or, if the API User is interested only in the network state change specific to CUSTOMER_ID 1, the API User can create a sieve with the repFilter attribute of CUSTOMER ID 1. An example of this request follows:

```
_cmd: CREATE
_obj_class: sieve
_attr: repFilter SS CUSTOMER_ID EQ I 1
_attr: eventFilter SS eventType EQ S
networkStateChange
<blank line>
```

The second half of the table "Attribute list and notification list table format" (page 34) (the notification list) contains three columns. The first column is the name of the event type, the second column contains the event information attributes that are conveyed as part of the notification, and the third column gives a description of the cause of the notification.

Note: In the tables, the abbreviation NA refers to Not Applicable.

Tables 2 to 12 describe the attributes and notifications of the object classes of the Network Model API.

Note: The node class info attributes are present if supported by the nodeType as indicated by the *nodeTypeId* attribute, and the link class info attributes are present if supported by the linkType as indicated by the *linkTypeId* attribute.

See the following tables for tables 2 to 12:

- "Network class attributes" (page 34)
- "TypeSet class attributes" (page 35)
- "Model class attributes and notifications" (page 35)
- "OrgNode class core attributes and notifications" (page 40)
- "OrgNode class core attributes and notifications" (page 40)
- "OrgLink class core attributes and notifications" (page 44)

- "Node class core and info attributes and notifications" (page 46)
- "Link class core and info attributes and notifications" (page 53)
- "NodeType class attributes" (page 56)
- "LinkType class attributes" (page 57)
- "AttrType class attributes" (page 57)

For a complete description of the sieve object class, see 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Table 1
Attribute list and notification list table format

Attributes			
Attribute	**Data type**	**Filter**	**Description**
Attribute name	Type of the attribute value	What operators can be used for filtering. Applies to GET and SIEVE filtering.	An explanation of attribute significance. The description column indicates if the attribute is the Object ID naming attribute.
Notifications			
Event type	Attributes		Description
Event name	The event information attributes		The notification cause

Table 2
Network class attributes

Attribute	**Data type**	**Filter**	**Description**
networkId	S	NA	The Object ID naming attribute. It is always orgRoot or compRoot.
objectClass	S	EQ,NE	Always network

Table 3
TypeSet class attributes

Attribute	Type	Filter	Description
objectClass	S	EQ,NE	Always typeSet
typeSetId	S	NA	The Object ID naming attribute. It can be nodeTypes, linkTypes, or attrTypes.

Table 4
Model class attributes and notifications

Attributes			
Attribute	**Data type**	**Filter**	**Description**
adminState	I	NA	Lock State of the model Two possible values: 0 for locked and 1 for unlocked. The model is locked whenever a new model is loaded. If the model is locked, the clients should not use the model until the model is unlocked.The model is unlocked when the model is active and accessible.
modelId	S	NA	The Object ID naming attribute. It is always networkModel.
modelName	S	NA	Name of the current model
modelPath	S	NA	Path where the instances files used to load the current model can be found
objectClass	S	NA	Always model
(Sheet 1 of 3)			

Table 4 (Continued)
Model class attributes and notifications

Attributes			
Attribute	**Data type**	**Filter**	**Description**
Notifications			
Event type	**Attributes**		**Description**
networkChange	netChangeType = nodePositionChange		A node has been moved to a new position. The event report contains information about the component name, the new X and Y coordinates and the name of the organization relative to which the position has changed.
	netChangeType = replaceBendpoints Change		A link bendpoints have changed. Bendpoints may have been removed, added, or changed. The event report contains information about the component name, the list of new X and Y coordinates of all the bendpoints and the name of the organization relative to which the bendpoints had changed.
	netChangeType = unlockModel		A Network Model client has unlocked the model The API considers that most of the information it knows is still valid, but it performs a reload to make sure the corrected information is captured. ***Note:*** The sieves remain even if the model has changed. Also, the API User may receive more unlocks than locks.

(Sheet 2 of 3)

Table 4 (Continued)
Model class attributes and notifications

Attributes		
Attribute	**Data type Filter**	**Description**
networkChange	netChangeType = startEdit	A Network Model edit session just started.
	netChangeType = endEdit	A Network Model edit session just terminated.
	netChangeType = lockModel	A Network Model client requested a lock on the model. A new model is instantiated. It is recommended that the API User not use the model until an unlock is received.
	netChangeType = unlockModel (no netChangeReas attribute)	A Network Model client has unlocked the model without making any changes. This notification can be received even if no prior lockModel notification was received.
	netChangeType = unlockModel netChangeReas = newModel	A Network Model client has unlocked the model after loading in a new model. It is recommended that none of the information currently known to the application be considered valid. ***Note:*** The sieves remain even if the model has changed. Also, the API User may receive more unlocks than locks.

(Sheet 3 of 3)

Table 5
Org class core attributes and notifications

Org class core attributes			
Attribute	**Data type**	**Filter**	**Description**
criticality	I	EQ,NE,LT,GT,GE, LE	This is an indication of the importance of the component. The range is 1 to 5.
networkState	E	EQ, NE	This is the state of the component based on its rawState and the states of the related components, subcomponents, and higher level components. Possible values are: INV, UNK, INSV, MTCE, MTCE_FROM_PARENT, ISTB, OOS, OSS_FROM_PARENT.
objectClass	S	EQ, NE	Always org
orgId	NI	EQ,NE,LEFT	The Object ID naming Attribute. An organization ID in the node format.
orgTypeId	S	EQ, NE	This is the type of the organization.
problemState	E	EQ, NE	This is the state of the problem maintained by the Expert Monitor Management (EMM) on that component. Possible values are: HAS_PROBLEM, PROBLEM_CLEAR, PR_UNKNOWN and PROBLEM_REMOVED.
rawState	E	EQ, NE	This is the state as received from the General Management Data Router (GMDR). Possible Values are: INV,UNK,INSV, ISTB,OOS.
stateSeverity	I	EQ, NE, LT,GT,GE,LE	This is an indication of the criticality of the fault. The combination of the networkState and stateSeverity gives an indication of the component trouble.The range is 1 to 5.
(Sheet 1 of 3)			

Table 5 (Continued)
Org class core attributes and notifications

Org class core attributes			
Attribute	**Data type**	**Filter**	**Description**
Org class info attributes			
Attribute	**Data type**	**Filter**	**Description**
MAP	S	NA	Path Name of background map from the Network Viewer

(Sheet 2 of 3)

Table 5 (Continued)
Org class core attributes and notifications

Org class core attributes			
Attribute	**Data type**	**Filter**	**Description**
Org class notifications			
Event type	**Attributes**		**Description**
networkChange	netChangeType = attributeChange		The MAP organization attributes have been changed. The event report contains the new map name.
	netChangeType = createNode		A new organization has been added to the model.
	netChangeType = deleteNode		An org has been removed from the model. Since the component is removed, none of its attributes are available. Therefore, repInfo and repFilter sieve attributes have no effect.
networkState Change	networkState		NetworkState for this node changed
	stateSeverity		The new networkState value and the state severity are sent.

(Sheet 3 of 3)

Table 6
OrgNode class core attributes and notifications

OrgNode class core attributes			
Attribute	**Data type**	**Filter**	**Description**
connNodes	NI Set of S	NA	IDs of the nodes connected to the component If more than one node is connected, each one is reported in a different attribute line.
connLinks	LI Set of S	NA	IDs of the links connected to the component. If more than one link is connected, each one is reported in a different attribute line.

(Sheet 1 of 4)

Table 6 (Continued)
OrgNode class core attributes and notifications

OrgNode class core attributes			
Attribute	**Data type**	**Filter**	**Description**
coordinates	S Set of S	NA	Coordinates of this node If the node exists in more than one organization, the coordinates in each organization are reported in different coordinate attribute lines. The format of each coordinate value is: <ORG>: <X> <Y>
criticality	I	EQ,NE,LT,GT, GE,LE	Indication of the importance of the component The range is 1 to 5.
networkState	E	EQ, NE	State of the component based on its rawState and the states of the related components, subcomponents, and higher level components Possible values are: INV, UNK, INSV, MTCE, MTCE_FROM_PARENT, ISTB, OOS, OSS_FROM_PARENT
objectClass	S	EQ, NE	Always node
orgNodeId	NI	EQ,NE,LEFT	The Object ID naming attribute. A component ID in the node format
orgNodeTypeId	S	EQ, NE	Type of organizational node
parents	NI Set of S	NA	IDs of the nodes that are the direct parents of this node Each parent is reported in a different parent attribute line.
problemState	E	EQ, NE	State of the problem maintained by the Expert Monitor Management (EMM) on that component Possible values are: HAS_PROBLEM, PROBLEM_CLEAR, PR_UNKNOWN, and PROBLEM_REMOVED

(Sheet 2 of 4)

Table 6 (Continued)
OrgNode class core attributes and notifications

OrgNode class core attributes			
Attribute	**Data type**	**Filter**	**Description**
rawState	E	EQ, NE	State as received from General Management Data Router (GMDR) Possible Values are: INV, UNK, INSV, ISTB, OOS
stateSeverity	I	EQ, NE, LT,GT,GE,LE	Criticality of the fault The combination of the networkState and stateSeverity gives an indication of the component trouble. The range is 1 to 5.
OrgNode class notifications			
Event type	**Attributes**		**Description**
networkState Change	networkState stateSeverity		NetworkState for this node changed The new network state value and the state severity are sent.
rawStateChange	rawState		RawState for this node changed The new raw state value is sent.
(Sheet 3 of 4)			

Table 6 (Continued)
OrgNode class core attributes and notifications

OrgNode class core attributes		
Attribute	**Data type Filter**	**Description**
Event type	**Attributes**	**Description**
networkChange	netChangeType = createNode	A new orgNode has been added to the model. Note that the notification is sometimes sent as soon as the orgNode is created but before it is actually given proper attributes (for example, orgNodes are usually created before they are positioned so their coordinates attribute is unreliable if extracted by a repInfo sieve attribute).
	netChangeType = deleteNode	An orgNode has been removed from the model. Since the component is removed, none of its attributes are available. Therefore, repInfo and repFilter sieve attributes have no effect.
	netChangeType = attributeChange	Some of the attributes have been changed. The event report contains the names of the changed attributes as well as their new values.
	netChangeType = reorgNode	An orgNode has been assigned to a new organizational node parent. Extracting the parent attribute with the sieve repInfo attribute yields the new parentage.
(Sheet 4 of 4)		

Table 7
OrgLink class core attributes and notifications

OrgLink class core attributes			
Attribute	**Data type**	**Filter**	**Description**
bendPoints	S Set of S	NA	If the link exists in more than one organization, the bendpoints in each organization are reported in different bendPoints attribute lines. The format of each bendpoint value is: <ORG>: <X1> <Y1> ...<Xn> <Yn>, where X1, Y1 are the coordinates of the bendpoint nearest the first endpoint node.
criticality	I	EQ, NE, LT,GT, GE,LE	The importance of the component The range is 1 to 5.
endPoints	NI Set of S	NA	IDs of the endpoints in the node format One endPoints attribute line exists for each endpoint.
networkState	E	EQ, NE	State of the component based on its rawState and the states of the related components, subcomponents, and higher level components Possible values are: INV, UNK, INSV, MTCE, MTCE_FROM_PARENT, ISTB, OOS, OSS_FROM_PARENT.
objectClass	S	EQ, NE	Always link
orgLinkId	LI	EQ,NE, LEFT	The Object ID naming attribute.
orgLinkTypeId	S	EQ, NE	Organizational link type
parents	NI, LI Set of S	NA	IDs of the nodes or links that are the direct parents of this node Each parent is reported in a different parent attribute line.
(Sheet 1 of 3)			

Table 7 (Continued)
OrgLink class core attributes and notifications

OrgLink class core attributes			
Attribute	**Data type**	**Filter**	**Description**
problemState	E	EQ, NE	State of the problem maintained by the Expert Monitor Management (EMM) on that component Possible values are: HAS_PROBLEM, PROBLEM_CLEAR, PR_UNKNOWN, and PROBLEM_REMOVED.
rawState	E	EQ, NE	State as received from the General Management Data Router (GMDR) Possible values are: INV,UNK,INSV, ISTB,OOS.
stateSeverity	I	EQ, NE, LT,GT, GE,LE	An indication of the criticality of the fault The combination of the networkState and stateSeverity gives an indication of the component trouble. The range is 1 to 5.
OrgLink class notifications			
Event types	**Attributes**		**Description**
networkChange	netChangeType = createLink		A new orgLink has been added to the model. Note that the notification is sent as soon as the link is created but before it is given bendpoints. This attribute is therefore unreliable if extracted by a repInfo sieve attribute.
	netChangeType = deleteLink		An orgLink has been removed from the model. Since the component is removed, none of its attributes are available. Therefore, repInfo and repFilter sieve attributes have no effect.
	netChangeType = reorgLink		An orgLink has been assigned to a new organizational link or node parent. Extracting the parent attribute with the sieve repInfo attribute yields the new parentage.

(Sheet 2 of 3)

Table 7 (Continued)
OrgLink class core attributes and notifications

OrgLink class core attributes			
Attribute	**Data type**	**Filter**	**Description**
networkState Change	networkState stateSeverity		The networkState for this link changed The new networkState value and the stateSeverity are sent.
rawStateChange	rawState		The rawState for this link changed The new rawState value is sent.
(Sheet 3 of 3)			

Table 8
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
ackLevel	E	EQ, NE	The acknowledgement level for the component. Possible values are: ACK_NONE (no maintenance or acknowledgement), ACK_TEMP (acknowledgement made), and ACK_PERM (maintenance or maintenance from parent state).
compId	NI	EQ,NE,LEFT	The Object ID naming attribute. A component ID in the node format
connLinks	LI Set of S	NA	IDs of the links connected to the component If more than one link is connected, each one is reported in a different attribute line in link format.
(Sheet 1 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
connNodes	NI Set of S	NA	IDs of the nodes connected to the component If more than one node is connected, each one is reported in a different attribute line in node format.
coordinates	S Set of S	NA	Coordinate of this node If the node exists in more than one organization, the coordinates in each organization are reported in different coordinates attribute lines. The format of each coordinates values is: <ORG>: <X> <Y>.
criticality	I	EQ,NE,LT, GT,GE,LE	The importance of the component The range is 1 to 5.
networkState	E	EQ, NE	The state of the component based on its rawState and the states of the related components, subcomponents, and higher level components Possible values are: INV, UNK, INSV, ACKED, MTCE, MTCE_FROM_PARENT, ISTB, OOS, OSS_FROM_PARENT.
nodeTypeId	S	EQ, NE	The type of node The possible value corresponds to an object in the nodeType containment hierarchy.
objectClass	S	EQ, NE	Always node
parents	NI Set of S	NA	IDs of the nodes that are the direct parents of this node Each parent is reported in a different parentNode attribute line in node format.
(Sheet 2 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
problemState	E	EQ, NE	The state of the problem maintained by the Expert Monitor Management (EMM) on that component Possible values are: HAS_PROBLEM, PROBLEM_CLEAR, PR_UNKNOWN, and PROBLEM_REMOVED.
rawState	E	EQ, NE	The state as received from the General Management Data Router (GMDR) Possible values are: INV,UNK,INSV, ISTB,OOS.
stateSeverity	I	EQ, NE, LT,GT,GE, LE	The criticality of the fault The combination of the networkState and stateSeverity gives an indication of the component trouble.The range is 1 to 5.
Node class info attributes (present if supported by the node *TypeId* attribute)			
Attribute	**Data type**	**Filter**	**Description**
ANID	I	EQ,NE,LT GT, GE,LE	ANID
CARD_TYPE	S	EQ,NE,LT, GT,GE,LE	The Passport card type
CURRENT_LP	S	EQ,NE,LT, GT,GE,LE	The Passport card's running Logical Processor (LP)
CUSTOMER_ID	I	EQ,NE,LT GT, GE,LE	The customer ID number of the Virtual Private Network (VPN)
GID	I	EQ,NE,LT GT, GE,LE	The Gateway ID
(Sheet 3 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
IMAGE	S	EQ,NE,LT GT, GE,LE	The image running on the component (PE)
INFO	S	EQ,NE,LT, GT,GE,LE	The component information
LINK_MODE	S	EQ,NE,LT, GT,GE,LE	The passport-DPN gateway link mode
LINK_TO_FRAMER	S	EQ,NE,LT, GT,GE,LE	On Passport interfaces, the Component ID of the related framer
LINK_TO_ INTERFACE	S	EQ,NE,LT, GT,GE,LE	On Passport framers, the link to the related interface
LOCATION	S	EQ,NE,LT GT, GE,LE	The module location
MAIN_CARD	S	EQ,NE,LT, GT,GE,LE	The Passport LPs main card component ID
MANAGING_OA	S	EQ,NE,LT, GT,GE,LE	The name of the Managing Operations Agent (OA) of the module
MAP	S	EQ,NE,LT, GT,GE,LE	The background map used for displaying the organization
MID	I	EQ,NE,LT GT, GE,LE	The module ID for the access module
NAMS_ID	I	EQ,NE,LT GT, GE,LE	The NAMS ID for the module
NMP	S	EQ,NE,LT, GT,GE,LE	The Network Management Protocol used
NODE_ID	I	EQ,NE,LT, GT,GE,LE	The passport node ID
(Sheet 4 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
PI_TYPE	S	EQ,NE,LT GT, GE,LE	The hardware type of PI
PROCESSOR_ TYPE	S	EQ,NE,LT GT, GE,LE	The hardware type of processor (PE)
RID	I	EQ,NE,LT GT, GE,LE	The routing ID for the module
RT_ZONE	I	EQ,NE,LT GT, GE,LE	The routing zone
SERVICES	S	EQ,NE,LT GT, GE,LE	The services provided by the component (PE)
SHELF_TYPE	S	EQ,NE,LT, GT,GE,LE	The module shelf type
SPARE_CARD	S	EQ,NE,LT, GT,GE,LE	The Passport Logical Processor (LP) spare card component ID
(Sheet 5 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
SWITCH_TYPE	S	EQ,NE,LT, GT,GE,LE	The switch type
(Sheet 6 of 8)			

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
Node class notifications			
Event type	**Attribute**		**Description**
ackLevelChange	ackLevel		The ackLevel state for this node has changed.
networkChange	netChangeType = createNode		A new node (module or subcomponent) added to the model Note that there is no such notification when a subcomponent is created by state propagation (SurNUp) -- one can rely on the state change notifications to be informed of these occurrences.
	netChangeType = deleteNode		A node has been removed from the model. Note that subcomponents removed through state propagation (surnup) are the object of such notifications. Since the component is removed, none of its attributes are available. Therefore, repInfo and repFilter sieve attributes have no effect.
	netChangeType = attributeChange		Some of the node attributes have been changed. The event report contains the names of the changed attributes as well as their new values.
	netChangeType = reorgNode		A node (module only) has been assigned to a new organizational node parent. Extracting the parent attribute with the sieve repInfo attribute yields the new parentage.

(Sheet 7 of 8)

Table 8 (Continued)
Node class core and info attributes and notifications

Node class core attributes			
Attribute	**Data type**	**Filter**	**Description**
networkState Change	networkState stateSeverity		The networkState for this node changed The new networkState value and the state severity are sent.
rawStateChange	rawState		This indicates that the rawState for this node has changed. The new rawState value is sent.
(Sheet 8 of 8)			

Table 9
Link class core and info attributes and notifications

Link class core attributes			
Attribute	**Data type**	**Filter**	**Description**
ackLevel	E	EQ, NE	The acknowledgement level for the component. Possible values are: ACK_NONE (no maintenance or acknowledgement), ACK_TEMP (acknowledgement made), and ACK_PERM (maintenance or maintenance from parent state).
bendPoints	S Set of S	NA	If the link exists in more than one organization, the bendPoints in each organization are reported in different bendPoint attribute lines. The format of each bendPoint value is: <ORG>: <X><Y> <X><Y>.
compId	LI	EQ,NE, LEFT	The Object ID naming attribute.
criticality	I	EQ, NE, LT,GT,GE, LE	The importance of the component The range is 1 to 5.
endPoints	NI Set of S	NA	The IDs of the endpoints in the node format. There is one endPoint attribute line for each endpoint.
(Sheet 1 of 3)			

Table 9 (Continued)
Link class core and info attributes and notifications

Link class core attributes			
Attribute	**Data type**	**Filter**	**Description**
linkTypeId	S	EQ, NE	The link type The possible value corresponds to an object in the linkType containment hierarchy.
networkState	E	EQ, NE	The state of the component based on its rawState and the states of the related components, subcomponents and higher level components The possible values are: INV, UNK, INSV, ACKED, MTCE, MTCE_FROM_PARENT, ISTB, OOS, OSS_FROM_PARENT.
objectClass	S	EQ, NE	Always link
parents	NI, LI Set of S	NA	The IDs of the nodes or links that are the direct parents of this node Each parent is reported in a different parent attribute line.
problemState	E	EQ, NE	The state of the problem maintained by the Expert Monitor Management (EMM) on that component The possible values are: HAS, CLEAR, UNKNOWN, REMOVED, and UNRECOGNIZED.
rawState	E	EQ, NE	The state as received from General Management Data Router (GMDR) The possible values are: INV,UNK,INSV, ISTB,OOS.
stateSeverity	I	EQ, NE, LT,GT,GE, LE	The criticality of the fault The combination of the networkState and stateSeverity gives an indication of the component trouble. The range is 1 to 5.
Link class info attributes (present if supported by the *linkTypeId* attribute)			
Attribute	**Data type**	**Filter**	**Description**
GID	I	NA	The Gateway ID
(Sheet 2 of 3)			

Table 9 (Continued)
Link class core and info attributes and notifications

Link class core attributes			
Attribute	**Data type**	**Filter**	**Description**
Link class notifications			
Event type	**Attribute**		**Description**
ackLevelChange	ackLevel		The ackLevel state for this node has changed.
network Change	netChangeType = createLink		A new link added to the model Note that the notification is sent as soon as the link is created, but before it is given bendpoints. Therefore, this attribute is unreliable if extracted by a repInfo sieve attribute.
	netChangeType = deleteLink		A link has been removed from the model. Since the component is removed, none of its attributes are available. Therefore, repInfo and repFilter sieve attributes have no effect.
	netChangeType = attributeChange		Some of the link attributes have been changed. The event report contains the names of the changed attributes as well as their new values
network Change	netChangeType = reorgLink		A link has been assigned to a new organizational link or node parent. Extracting the parent attribute with the sieve repInfo attribute yields the new parentage.
networkStateChange	networkState stateSeverity		The networkState for this link changed The new networkState value and the state severity are sent.
rawState Change	rawState		The rawState for this link changed The new rawState value is sent.

(Sheet 3 of 3)

Table 10
NodeType class attributes

Attribute	Data type	Filter	Description
connLinkTypes	S SetOfS	NA	The possible types of links that can be connected to this type of node The value corresponds to an object in the linkType containment hierarchy. If more than one connected linkType exists, then each one of them is reported in a separate connLinkTypes attribute line.
legalName	S	NA	The legal name value for this node in Grep syntax
nodeTypeId	S	NA	The Object ID naming attribute.
objectClass	S	NA	Always nodeType
parentNodeTypes	S SetOfS	NA	The possible immediate parent nodeType for this nodeType The value corresponds to an object in the nodeType containment hierarchy. If more than one parent nodeType exists, then each one of them is reported in a separate parentNodeTypes attribute line.
possAttrTypeIds	S SetOfS	NA	The possible info attribute names for this node type The value corresponds to an object in the attrType containment hierarchy. If more than one info attribute is defined for the nodeType, each one of them is reported in a separate possAttrTypeIds attribute line.

Table 11
LinkType class attributes

Attribute	Data type	Filter	Description
legEndpts	S SetOfS	NA	The possible pairs of nodeTypes connected to this linkType The value corresponds to two objects in the nodeType containment hierarchy. If more than one pair of node types exist, then each one of them is reported in a separate legEndpts attribute line. The format is: <nodeType>:<nodeType>.
linkTypeId	S	NA	The Object ID naming attribute.
objectClass	S	NA	Always linkType
possAttrTypeIds	S SetOfS	NA	The possible info attribute names for this linkType The value corresponds to an object in the attrType containment hierarchy. If more than one info attribute is defined for the nodeType, each one of them is reported in a separate possTypeIds attribute line.

Table 12
AttrType class attributes

Attribute	Data type	Filter	Description
attrTypeId	S	NA	The Object ID naming attribute.
defaultValue	S, I	NA	The default value for this infoAttribute The defaultValue is only reported if it exists.
explanation	S	NA	The significance of this infoAttribute
(Sheet 1 of 2)			

Table 12 (Continued)
AttrType class attributes

Attribute	Data type	Filter	Description
objectClass	S	NA	Always infoAttribute.
possValues	S, RI SetOfS	NA	The possible values for this infoAttribute If more than one value is possible, they are reported on different possValues attribute lines. If the possible value is an integer range, this value is reported on a single line and RI is used as a type specifier.

(Sheet 2 of 2)

Chapter 3
Using the Network Model API

This section explains how to install and use the Network Model API. This section contains the following information:

Code conventions

There are two 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 octothorp (#) is treated as a comment.

Installing and configuring the Network Model API

The Network Model API software requires no additional installation or configuration procedures. See 241-6001-100 *Preside MDM Installation* for details on installing the Network Model software, and 241-6001-015 *Preside MDM Network Model Administrator Guide* for details on configuring the Network Model software.

Starting a session with the Network Model API

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

```
/opt/MagellanNMS/bin/nmapi
```

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

```
_version: x.y Network Model API
```

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

Command line options

The API User may want to send the Network Model API an input file containing many requests, and then send the responses to an output file. The API User may also want to treat the errors differently than the response records. This is achieved by using the command line options.

```
/opt/MagellanNMS/bin/nmapi[-o <output filename> |
+o <output filename>]
[-f <input file name>] \
[-width <output width> | -w <output \
width>] [-help]\ [-h <host name>] \
[-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 standard output stream (*stdout*).

`-f <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.

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

`-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; then the 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.

Shortcut command line options

The Network Model API has the following shortcut command line options.

```
/opt/MagellanNMS/bin/nmapi[-net|-org|-node|-onode|
-link| \-olink <name>]
[-scope <base |all | next>]
```

```
[-attr_id <all | <name> >]
[-filt <attr> <op> <type> <value>]
[-sieve rstate | nstate | nchange]
```

where:

`-net <name>` selects a network object

`-org <name>` selects an org object

`-onode <name>` selects an orgNode object

`-link <name>` selects a link object

`-olink <name>` selects an orgLink object

`-scope <base |all | next>` sets the options for the *_scope*: line for targeting the object

`-attr_id <all | <name>>` selects what attributes are reported

`-filt <attr> <op> <type> <value>` sets the standard fields for the filter for the request

`-sieve rstate | nstate | nchange` sets the options for sieve creation; rawStateChange notification, networkStateChange notification, and networkChange notification. If none are specified, all are reported.

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: If an interactive session is used, it is recommended that you use the -v option (verbose mode).

Example

```
nmapi [-h <host name>] [-v]
```

where:

`-h <host name>` indicates the host name.

`-v` selects verbose mode.

The response is:

```
8.0 Network Model API copyright: 1991-1994 Nortel
Networks Corporation.
Connected to API Server "pdo_service" on "bcars561".
_version: 8.0 Network Model API
```

Input through an input file name

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

```
_cmd: get
_obj_class: node
_obj_id: compId NI PM R73
_scope: base
```

Example

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

The response is:

```
_version: 8.0 Network Model API
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
<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/nmapi -h <host name>
```

The response is:

```
_version: 8.0 Network Model API
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
<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/nmapi -h <host name> < toto
```

The response is:

```
_version: 8.0 Network Model API
<blank line>
_obj_class: node
_obj_id: compId NI PM R73
<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.
```

Using shortcuts

Using the -node/-onode/-link/-olink option along with -scope/-attr, you can invoke a GET query on a node, organizational node, link, or organizational link with the specified scope and attribute selections. You can also use the -sieve option to automatically create a sieve of the specified type. The standard Provider options are also supported. If none of the shortcut options are used, the Network Model API Provider acts just like the Generic Provider but configured explicitly to communicate with the Network Model API Server. The following examples demonstrate the use of the Generic API Provider through the Network Model API Provider.

Plain invocation of the Provider for Network Model Access (using the -node shortcut):

```
$ /opt/MagellanNMS/bin/nmapi -node "PM C01"
_version: 8.0 Network Model API
<blank line>
_obj_class: node
_obj_id: compId NI PM C01
<blank line>
_end_resp: GET
_time: 1993 10 08 18 46 30
<blank line>
_end: USER_REQUEST
<blank line>
```

Using the -v (verbose) option (using the -node shortcut):

```
$ /opt/MagellanNMS/bin/nmapi -node "PM C01" -v
<blank line>
Network Model API 8.0 copyright: 1991, 1992, 1993
Nortel Networks Corporation.
Connected to API Server "pdo_service" on "localhost".
<blank line>
version: 8.0 Network Model API
<blank line>
_obj_class: node
_obj_id: compId NI PM C01
<blank line>
_end_resp: GET
_time: 1993 10 08 18 59 37
<blank line>
_end: USER_REQUEST
<blank line>
Exiting API Provider with 0 status.
```

Terminating Network Model API access

To terminate the Network Model 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).

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

The Network Model API supports the following commands:

- GET
- CREATE sieves and event reports

- SET
- DELETE

The following sections describe the Network Model API support of these commands. The source code for the subsequent examples is found in the file */opt/MagellanNMS/bin/nmapi_examples*.

The GET command

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

The GET request retrieves node information and sieve definitions as shown in the following examples. These sections contain many examples illustrating the GET service requests. They do not, however, contain examples for each possible combination of object, attribute, service, scope, and filter. The examples are intended instead to cover the most common and useful combinations.

Using scope to get all the organizations

To retrieve all the organizations, issue the following request:

Example

```
_cmd: GET
_obj_class: network
_obj_id: networkId S orgRoot
_scope: NEXT
<blank line>
```

A typical response message is:

```
_obj_class: org
_obj_id: orgId NI DEFAULT ALL
<blank line>
_obj_class: org
_obj_id: orgId NI DPN R3-ORG
<blank line>
_obj_class: org
_obj_id: orgId NI DPN R4-ORG
<blank line>
```

```
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

where each record represents a different top level organizational node.

Using scope and filter to get all the Packet Modules

To retrieve all the Packet Module (PM) components, issue either of the following requests:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: NEXT
_filter: nodeTypeId EQ S PM
<blank line>
```

or

```
_cmd: GET
_obj_class: orgNode
_obj_id: orgNodeId NI DEFAULT_SITE ALL
_scope: NEXT
_filter: nodeTypeId EQ S PM
<blank line
```

A typical response message is:

```
_obj_class: node
_obj_id: compId NI PM SO1
<blank line>
_obj_class: node
_obj_id: compId NI PM SO2
<blank line>
_obj_class: node
_obj_id: compId NI PM SO3
<blank line>
...
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Note: The API User can use either the network object *compRoot* or the orgNode *DEFAULT_SITE ALL* with the scope *NEXT* as the base object to get all the module level components.

The orgNode *DEFAULT_SITE ALL* is always present in each of the Network Models and can be used as a base node when it is required to target physical objects of the Network Model.

Using scope and filter to get all the physical links

To retrieve all the physical links, issue either of the following requests:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: NEXT
_filter: objectClass EQ S link
<blank line>
```

or

```
_cmd: GET
_obj_class: orgNode
_obj_id: orgNodeId NI DEFAULT_SITE ALL
_scope: NEXT
_filter: objectClass EQ S link
<blank line>
```

A typical response message is:

```
_obj_class: link
_obj_id: compId LI DL:PM C02 PE 1 PI 1 PO 1:DIU 1:
<blank line>
_obj_class: link
_obj_id: compId LI GW:NM C03 LINE 1 SCAN 3 HSLC
1:FNMOD\
EUROPE:
<blank line>
_obj_class: link
_obj_id: compId LI NL:NM C03 LINE 5 SCAN 1 LC 1:PM E02\
PE 1 PI 1 PO 1:
<blank line>
...
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using scope and filter to get all the physical links of a particular type

To retrieve all the physical links of a particular type, issue either of the following requests:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: NEXT
_filter: linkTypeId EQ S TK
<blank line>
```

or

```
_cmd: GET
_obj_class: orgNode
_obj_id: orgNodeId NI DEFAULT_SITE ALL
_scope: NEXT
_filter: linkTypeId EQ S TK
<blank line>
```

A typical response message is:

```
_obj_class: link
_obj_id: compId LI TK:NM C03 TRNK 8:NM G02 TRNK 17:
<blank line>
_obj_class: link
_obj_id: compId LI TK:PM G01 PE 1 PI 1 PO 1:PM S02 PE 1\
PI 1 PO 1:
<blank line>
...
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using the *connLink* attribute to get all the links connected to a specific endpoint

To retrieve all the links connected to a specific endpoint, PM S01 PE 1 PI 1 PO 1, issue the following request:

```
_cmd: GET
_obj_class: node
_obj_id: compId NI PM S01 PE 1 PI 1 PO 1
_attr_id: connLinks
<blank line>
```

A typical response message is:

```
_obj_class: node
_obj_id: compId NI PM S01 PE 1 PI 1 PO 1
_attr: connLinks LI NL:PM S01 PE 1 PI 1 PO 1:PM S02 PE
1\
PI 1 PO 2:
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using the *attr_id* ALL to get all the attribute values (core and info) of a specific node

To retrieve all the attribute values of a specific node, PM R3, issue the following request:

```
_cmd: GET
_obj_class: node
_obj_id: compId NI PM R3
_attr_id: ALL
<blank line>
```

A typical response message is:

```
_obj_class: node
_obj_id: compId NI PM R3
_attr: networkState E UNK
_attr: stateSeverity I 0
_attr: rawState E UNK
_attr: problemState E REMOVED
_attr: nodeTypeId S PM
_attr: criticality I 4
_attr: coordinates S DEFAULT ALL: 50 50
_attr: coordinates S PASSPORT: 50 50
_attr: coordinates S ILOOK CLUSTER: 66 262
_attr: parents NI NEW_SITE DPN_PASSPORT
_attr: parents NI SITE CORENET
_attr: parents NI DEFAULT_SITE ALL
_attr: connNodes NI PM R3 PE 3 PI 3 PO 1
_attr: connNodes NI PM R3 PE 4 PI 4 PO 1
_attr: connLinks LI TK:PM R3 PE 3 PI 3 PO 1:\
PM R72 PE 27 PI 27 PO 1:
_attr: connLinks LI TK:PM R3 PE 4 PI 4 PO 1:\
PM R70 PE 22 PI 22 PO 1:
```

```
_attr: CUSTOMER_ID I 0
_attr: MANAGING_OA S CORENCS
_attr: MID I 403
_attr: NAMS_ID I 4003
_attr: RID I 3
_attr: RT_ZONE I 1
_attr: SHELF_TYPE S DUAL_SHELF
_attr: SWITCH_TYPE S RM
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using multiple filters

To retrieve all the PMs in service and whose criticality is 5, issue either of the following requests:

```
_cmd: GET
_obj_class: network
_obj_id: networkId S compRoot
_scope: NEXT
_filter: criticality EQ I 5
_filter: networkState EQ E INSV
_filter: nodeTypeId EQ S PM
<blank line>
```

or

```
_cmd: GET
_obj_class: orgNode
_obj_id: orgNodeId NI DEFAULT_SITE ALL
_scope: NEXT
_filter: criticality EQ I 5
_filter: networkState EQ E INSV
_filter: nodeTypeId EQ S PM
<blank line>
```

A typical response message is:

```
_obj_class: node
_obj_id: compId NI DMSBUS BUS01
<blank line>
...
```

```
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using scope to get all the linkTypes

To retrieve all the possible linkTypes, issue the following request:

```
_cmd: GET
_obj_class: typeSet
_obj_id: typeSetId S linkTypes
_scope: NEXT
<blank line>
```

A typical response message is:

```
_obj_class: linkType
_obj_id: linkTypeId S CL
<blank line>
_obj_class: linkType
_obj_id: linkTypeId S DL
<blank line>
_obj_class: linkType
_obj_id:linkTypeId GW
<blank line>
...
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using scope to get the nodeTypes

To retrieve all the possible nodeTypes that can be children of a PM-PI, issue the following request:

```
_cmd: GET
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PI
_scope: NEXT
<blank line>
```

A typical response message is:

```
_obj_class: nodeType
_obj_id: nodeTypeId S PM-DISK
<blank line>
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PO
```

```
<blank line>
...
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using the *connLinkTypes* attribute to get the set of link types

To retrieve the set of link types that can be connected to a PM-PO node type, issue the following request:

```
_cmd: GET
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PO
_attr_id: connLinkTypes
<blank line>
```

A typical response message is:

```
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PO
_attr: connLinkTypes S CL
_attr: connLinkTypes S DBNL
_attr: connLinkTypes S DL
_attr: connLinkTypes S DNL
_attr: connLinkTypes S GW
_attr: connLinkTypes S LL
_attr: connLinkTypes S NL
_attr: connLinkTypes S TK
_attr: connLinkTypes S WL
_attr: connLinkTypes S XL
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Retrieving all the possible info attributes of a PM-PO

To retrieve all the possible info attributes of a PM-PO node type, issue the following request:

```
_cmd: GET
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PO
_attr_id: ALL
<blank line>
```

A typical response message is:

```
_obj_class: nodeType
_obj_id: nodeTypeId S PM-PO
_attr: parentNodeTypes S PM-PI
_attr: connLinkTypes S CL
_attr: connLinkTypes S DBNL
_attr: connLinkTypes S DL
_attr: connLinkTypes S DNL
_attr: connLinkTypes S GW
_attr: connLinkTypes S LL
_attr: connLinkTypes S NL
_attr: connLinkTypes S TK
_attr: connLinkTypes S WL
_attr: connLinkTypes S XL
_attr: possAtrrTypes S CUSTOMER_ID
_attr: possAttrTypes S INFO
_attr: possAttrTypes S LINK_TYPE
_attr: legalNames S ^[1-9]$
_attr: legalNames S ^[1][0-9]$
_attr: legalNames S ^[2][0-9]$
_attr: legalNames S ^[3][0-9]$
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Using the *legEndpts* attribute to determine the possible node type pairs

To retrieve all the possible node type pairs that can be connected to a link type GW, issue the following request:

```
_cmd: GET
_obj_class: linkType
_obj_id: linkTypeId S GW
_attr_id: legEndpts
<blank line>
```

A typical response message is:

```
_obj_class: linkType
_obj_id: linkTypeId S GW
_attr: legEndpts S PM-PO:FNMOD
_attr: legEndpts S NM-LC:FNMOD
_attr: legEndpts S NM-HSLC:FNMOD
```

```
_attr: legEndpts S HOST:FNMOD
_attr: legEndpts S BB:FNMOD
_attr: legEndpts S BB:HOST
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Retrieving all the possible values for the *PROCESSOR_TYPE* attribute

To retrieve all the possible values for the *PROCESSOR_TYPE* attribute, issue the following request:

```
_cmd: GET
_obj_class: attrType
_obj_id: attrTypeId S PROCESSOR_TYPE
_attr_id: possValues
<blank line>
```

A typical response message is:

```
_obj_class: attrType
_obj_id: attrTypeId S PROCESSOR_TYPE
_attr: possValues S PE286
_attr: possValues S PE386
_attr: possValues S PE88
<blank line>
_end_resp: GET
_time: 1994 02 05 18 59 02
<blank line>
```

Creating sieves and event reports

The CREATE command creates sieves to receive the following types of event reports:

- networkChange
- networkStateChange
- rawStateChange

A CREATE request results in one successful response message or an error message followed by an end-of-response. The successful response message contains the sieve object identifier.

Creating a sieve to receive *rawStateChange* and *networkStateChange* reports

To receive all the rawStateChange and networkStateChange reports, issue the following request:

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

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
<blank line>
_end_resp: CREATE
_time: 1994 02 10 10 44 22
<blank line>
```

The typical event reports are:

```
_sieve_id: 1
_obj_class: node
_obj_id: compId NI PM AC7043
_event_type: networkStateChange
_time: 1994 02 10 10 47 13
_attr: networkState E OOS
_attr: stateSeverity I 4
<blank line>
_sieve_id: 1
_obj_class: node
_obj_id: compId NI PM AC7043
_event_type: rawStateChange
_time: 1994 02 10 10 47 22
_attr: rawState E OOS
<blank line>
...
```

Creating a sieve to receive *rawStateChange* and *networkStateChange* reports for all modules under a specific site

To receive all the rawStateChange and networkStateChange reports for all the modules under the SITE BARKER, issue the following requests:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S
networkStateChange
_attr: eventFilter SS eventType EQ S rawStateChange
_attr: repOClass S orgNode
_attr: repOid SS orgNodeId NI SITE BARKER
_attr: repScope E NEXT
_attr: repFilter SS objectClass NE S link
<blank line>
```

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
<blank line>
_end_resp: CREATE
_time: 1994 02 10 10 44 22
<blank line>
```

The typical event reports are:

```
_sieve_id: 1
_obj_class: node
_obj_id: compId NI EM NODEY219
_event_type: rawStateChange
_time: 1994 02 10 10 51 04
_attr: rawState E UNK
<blank line>
_sieve_id: 1
_obj_class: node
_obj_id: compId NI EM NODEY219
_event_type: networkStateChange
_time: 1994 02 10 10 51 09
_attr: networkState E INSV
_attr: stateSeverity I 0
<blank line>
...
```

Creating a sieve to receive *rawStateChange* and *networkStateChange* reports for a specific module and all its subcomponents

To receive all the rawStateChange and networkStateChange reports for all the module PM R8 and all its subcomponents, issue one of the following requests:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S
networkStateChange
_attr: eventFilter SS eventType EQ S rawStateChange
_attr: repFilter SS compId LEFT NI PM R8*
_attr: repFilter SS compId EQ NI PM R8
<blank line>
```

Note: The * character represents a blank character and is required to filter out components such as R81, R82, R83 ... R89.

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS eventType EQ S
networkStateChange
_attr: eventFilter SS eventType EQ S rawStateChange
_attr: repOClass S node
_attr: repOid SS compId NI PM R8
_attr: repScope E ALL
<blank line>
```

The typical response message is:

```
_obj_class: sieve
_obj_id: sieveId I 1
<blank line>
_end_resp: CREATE
_time: 1994 02 10 10 44 22
<blank line>
```

The typical event reports are:

```
_sieve_id: 1
_obj_class: node
_obj_id: compId NI PM R8 PE 1
_event_type: networkStateChange
_time: 1994 02 10 11 03 22
```

```
_attr: networkState E OOS
_attr: stateSeverity I 0
<blank line>
...
```

and

```
_sieve_id: 1
_obj_class: node
_obj_id: compId NI PM R8 PE 12
_event_type: rawStateChange
_time: 1994 02 10 11 03 25
_attr: rawState E OOS
<blank line>
...
```

Setting the sieve

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. The only attribute that can be changed is the *admState* attribute. In the following 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>
```

A 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.

Deleting the sieve

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.

The DELETE request removes a sieve. For example, to delete a sieve having an ID of 42, the following request is issued:

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

A 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>
```

Create sieve templates

In the following templates, mandatory lines are preceded by an exclamation (!) character.

NetworkstateChange events

The template for the networkStateChange event is as follows:

```
! _cmd: CREATE
! _obj_class: sieve
! _attr: eventFilter SS eventType EQ S networkState
Change
_attr: eventFilter SS objectClass <EQ | NE>S<node |link
|
orgLink | orgNode | org>

_attr: eventFilter SS networkState <EQ | NE > E <INV |
UNK
| INSV | MTCE| MTCE_FROM_PARENT| ISTB | OOS |
OOS_FROM_PARENT>
_attr: eventFilter SS stateSeverity <EQ |
NE|LT|GT|GE|LE> I <value between 1 and 5>
```

```
_attr: eventFilter SS < compId | orgLinkId | orgNodeId
| orgId> <EQ|NE|LEFT> < NI| LI|S> <value>
_attr: repOClass S <node | link | orgLink | orgNode |
org|network>
_attr: repOid SS <compId | orgLinkId | orgNodeId |orgId
|networkId> <NI |LI | S> <objectName>
_attr: repScope E <base | next | all >
_attr: repFilter SS <attribute name> <allowedoperator>
<attribute value type> <value>
_attr: repInfo S <attribute name | ALL>
_attr: eventInfo S < networkState | stateSeverity>
```

Note: For attribute *repFilter SS*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. For this attribute and **<***allowedoperator***>** and **<***attribute value type***>**, see the tables in "Object model" (page 21).

Note: For attribute *repInfo S*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. See the tables in "Object model" (page 21).

NetworkChange events

The template for the networkChange event is as follows:

```
! _cmd: CREATE
! _obj_class: sieve
! _attr: eventFilter SS eventType EQ S networkChange
_attr: eventFilter SS objectClass <EQ | NE> S <node |
link | orgLink | orgNode | org| model|>
_attr: netChangeType <EQ | NE > S <createNode |
deleteNode| reorgNode | createLink | deleteLink |
reorgLink | lockModel | unlockModel |
replaceBendpointsChange | attributeChange |
nodePositionChange | startEdit | endEdit >
_attr: netChangeReason <EQ | NE > S <newModel |
modelIncrement>
_attr: eventFilter SS < compId | orgLinkId | orgNodeId
| orgId> <EQ|NE|LEFT> < NI| LI|S> <value>
_attr: repOClass S <node | link | orgLink | orgNode |
org|network| model>
_attr: repOid SS <compId | orgLinkId | orgNodeId |orgId
|networkId | modelId> <NI |LI | S> <objectName>
```

```
_attr: repScope E <base | next | all >
_attr: repFilter SS <attribute name>
<allowed operator> <attribute value type> <value>
_attr: repInfo S <attribute name | ALL>
_attr: eventInfo S < netChangeType|netChangeReason>
```

Note: If the netChangeType value is deleteLink or deleteNode the repScope, repOid, and repOClass values are ignored.

Note: For attribute *repFilter SS*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. For this attribute and <*allowed operator*> and <*attribute value type*>, see the tables in "Object model" (page 21).

Note: For attribute *repInfo S*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. See the tables in "Object model" (page 21).

RawStateChange events

The template for the rawStateChange event is as follows:

```
! _cmd: CREATE
! _obj_class: sieve
! _attr: eventFilter SS eventType EQ S rawStateChange
_attr: eventFilter SS objectClass <EQ | NE> S <node |
link | orgLink | orgNode | org>
_attr: eventFilter SS rawState <EQ | NE > E <INV | UNK
| INSV | ISTB | OOS>
_attr: eventFilter SS < compId | orgLinkId | orgNodeId
| orgId> <EQ|NE|LEFT> < NI| LI|S> <value>
_attr: repOClass S <node | link | orgLink | orgNode |
org|network>
_attr: repOid SS <compId | orgLinkId | orgNodeId |orgId
|networkId> <NI |LI | S> <objectName>
_attr: repScope E <base | next | all >
_attr: repFilter SS <attribute name>
<allowed operator> <attribute value type> <value>
_attr: repInfo S <attribute name


| ALL>
_attr: eventInfo S < rawState >
```

Note: For attribute *repFilter SS*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. For this attribute and <*allowed operator*> and <*attribute value type*>, see the tables in "Object model" (page 21).

Note: For attribute *repInfo S*, the attribute name is the name of the attribute of the reporting object for which filtering is allowed. See the tables in "Object model" (page 21).

Optimization

The Network Model API does not optimize the queries of the API User. Therefore the API User must be careful when selecting the set of targeted objects.

Example

To efficiently extract information about all the PMs, start at *compRoot*

and use scope NEXT. Starting at *orgRoot* and using scope ALL with a filter on *nodeTypeId* PM is inefficient.

Appendix A
Compliance statement

This appendix indicates how the Preside Multiservice Data Manager (MDM) Network Model API conforms to the general API functionality as described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

This appendix contains the following information:

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 86) indicates the messages that are supported by the Network Model 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	N
ACTION response	send	N
CREATE request	receive	C[3]
CREATE response	send	Y
DELETE request	receive	C[4]
DELETE response	send	Y
REGISTER request	receive	N
REGISTER response	send	N
DEREGISTER request	receive	N
DEREGISTER response	send	N
EVENT-REPORT message	send	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)		

1 The GET command is supported on every object except the root object. The root object is not addressable, and therefore cannot be used as a basis for a scoped request. Filtering restrictions are class dependant. Refer to "Object model" (page 21)). Access to the sieve information is restricted to the sieve creators.

2 The SET command is supported for the sieve object. Scoping and filtering are not supported. The ADD, DEL, and DEF modification types are not supported.

3 The CREATE command is supported for the sieve object. The *obj_id*, *sup_obj_id*, and *ref_obj_id* attributes are not supported. They are ignored if they are provided in the CREATE request.

4 The DELETE command is supported for the sieve object. Scoping and filtering are not supported.

5 More than one type of event may be specified for each sieve. If no type of event is specified, then the resulting sieve reports all the possible notifications. The repFilter is subject to the same restrictions as the GET filtering (refer to "Object model" (page 21)).

API message lines

The table "API message lines" (page 88) indicates the message lines and keywords that are supported by the Network Model API. It describes overall support for the message line. The table "Optional message lines" (page 91) indicates the optional message lines that are supported by the Network Model 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		N
_attr		Y
_attr_id	general support ALL	Y Y
_block		Y
_capability		N
_cmd	REGISTER, DEREGISTER, ACTION, CREATE, SET, DELETE, GET	Y (see also the table “API message types” (page 86))
_echo		N
_end	general support USER_REQUEST FATAL_ERROR SYSTEM_SHUTDOWN LOST_CONNECTION LOST_SESSION	Y Y Y Y Y Y
_end_block		Y
(Sheet 1 of 3)		

Table 14 (Continued)
API message lines

Message line	Keywords	Supported
_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
_filter	general support P EQ NE LT GT LE GE LEFT MIDDLE RIGHT IN NOTIN	C[1] N C[1] C[1] C[1] C[1] C[1] C[1] C[1] N N N N
_inv_id		Y
_mod	general support ADD DEL DEF REP	C [2] N N N Y
(Sheet 2 of 3)		

Table 14 (Continued)
API message lines

Message line	Keywords	Supported
_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		N
_user_id		Y
_version		Y
(Sheet 3 of 3)		

1 The GET command is supported on every object except the root object. The root object is not addressable, and therefore cannot be used as a basis for a scoped request. Filtering restrictions are class dependant. (Refer to “Object model” (page 21)). Access to the sieve information is restricted to the sieve creators.

2 The SET command is supported for the sieve object. Scoping and filtering are 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	N N N N
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	N N N
DELETE response	_inv_id	N
REGISTER request	_inv_id _user_id _password _attr	N N N N

(Sheet 1 of 2)

Table 15 (Continued)
Optional message lines

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

1 The SET command is supported for the sieve object. Scoping and filtering are 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 92) indicates the API data types supported by the Network Model 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)	N
I (Integer)	Y
H (Hexadecimal)	N
(Sheet 1 of 2)	

Table 16 (Continued)
API data types

API data type	Supported
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)	N
SB (Sequence of Booleans)	N
RS (Range of Strings)	N
RI (Range of Integers)	Y
(Sheet 2 of 2)	

Sieve object support

The table "Sieve object attributes" (page 93) indicates the attributes of the sieve object that are supported by the Network Model API.

Table 17
Sieve object attributes

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

Table 17 (Continued)
Sieve object attributes

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

1 The repFilter is subject to the same restrictions as the GET filtering (refer to “Object model” (page 21)).

Appendix B
Error messages

As well as the errors documented in 241-6001-200 *Preside MDM Application Programming Interface Primer*, the Network Model API can produce the errors listed in this appendix. This appendix contains the following information:

Error summary

The table "Error summary" (page 95) is a summary of error messages.

Table 18
Error summary

Error message
NO_SUCH_OBJECT_ID base object not found
INVALID_ATTRIBUTE_VALUE invalid operator in objectClass repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in objectClass repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in objectClass repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in compId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in compId repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in compId repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgId repFilter
(Sheet 1 of 6)

Table 18 (Continued)
Error summary

Error message
INVALID_ATTRIBUTE_VALUE no attribute value in orgId repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgLinkId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgLinkId repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in orgLinkId repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgNodeId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgNodeId repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in orgNodeId repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in networkId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in networkId repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in networkId repFilter
INVALID_ATTRIBUTE_VALUE invalid operator in modelId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in modelId repFilter
INVALID_ATTRIBUTE_VALUE no attribute value in modelId repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute in repFilter
INVALID_ATTRIBUTE_VALUE invalid attribute in eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in objectClass eventFilter
INVALID_ATTRIBUTE_VALUE no attribute value in objectClass eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in compId eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in compId eventFilter
INVALID_ATTRIBUTE_VALUE no attribute value in compId eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgId eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgId eventFilter
INVALID_ATTRIBUTE_VALUE no attribute value in orgId eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgLinkId eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgLinkId eventFilter
(Sheet 2 of 6)

Table 18 (Continued)
Error summary

Error message
INVALID_ATTRIBUTE_VALUE no attribute value in orgLinkId eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in orgNodeId eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in orgNodeId eventFilter
INVALID_ATTRIBUTE_VALUE no attribute value in orgNodeId eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in networkId eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in networkId eventFilter
INVALID_ATTRIBUTE_VALUE no attribute value in networkId eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in modelId eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in eventType eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in eventType eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in eventType eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in networkState eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in networkState eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in networkState eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in rawState eventFilter
NVALID_ATTRIBUTE_VALUE invalid attribute type in rawState eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in rawState eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in stateSeverity eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in stateSeverity eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in rawState eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in stateSeverity eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in stateSeverity eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in stateSeverity eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in netChangeType eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in netChangeType eventFilter
(Sheet 3 of 6)

Table 18 (Continued)
Error summary

Error message
INVALID_ATTRIBUTE_VALUE invalid operator in netChangeType eventFilter
INVALID_ATTRIBUTE_VALUE invalid value in netChangeReas eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in netChangeReas eventFilter
INVALID_ATTRIBUTE_VALUE invalid operator in networkChangeReas eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute in eventFilter
INVALID_ATTRIBUTE_VALUE invalid attribute type in eventInfo
INVALID_ATTRIBUTE_VALUE invalid attribute value in eventInfo
SYNTAX_ERROR unknown reporting object id (repOid)
SYNTAX_ERROR unknown reporting object type (repOClass)
INVALID_ATTRIBUTE_VALUE invalid attribute type in eventInfo
INVALID_ATTRIBUTE_VALUE invalid reporting object type
INVALID_ATTRIBUTE_VALUE invalid repOid value for supplied repOClass
INVALID_ATTRIBUTE_VALUE invalid repOid value for supplied repOClass
INVALID_ATTRIBUTE_VALUE invalid value in eventType eventFilter
SYNTAX_ERROR invalid attribute type in obj_id
SYNTAX_ERROR invalid attribute name in obj_id
SYNTAX_ERROR no obj_class supplied
SYNTAX_ERROR no obj_id supplied
NOT_SUPPORTED set for specified obj_class
INVALID_OBJECT_ID no value
INVALID_OBJECT_ID no naming attribute value
INVALID_ATTRIBUTE_NAME bad sieve attribute
INVALID_ATTRIBUTE_NAME invalid attribute in GET sieve
INVALID_OBJECT_TYPE invalid object type
NOT_SUPPORTED create for specified obj_class
(Sheet 4 of 6)

Table 18 (Continued)
Error summary

Error message
NOT_SUPPORTED obj_id specification in create sieve
NOT_SUPPORTED register command
NOT_SUPPORTED deregister command
SYNTAX_ERROR no command
SYNTAX_ERROR invalid API Command
SYNTAX_ERROR no target reporting object supplied for repScope
INVALID_ATTRIBUTE_VALUE invalid repScope value
APPLICATION_ERROR service unavailable not connected to nmc
APPLICATION_ERROR service unavailable no network model
APPLICATION_ERROR service unavailable network model is locked
SYNTAX_ERROR duplication of annotation attribute
SYNTAX_ERROR duplication of admState attribute
SYNTAX_ERROR invalid attribute type for admState
SYNTAX_ERROR invalid value for admState
APPLICATION_ERROR maximum number of sieves reached
NO_SUCH_OBJECT_ID No matching Sieve
SYNTAX_ERROR invalid duplication of Command line
SYNTAX_ERROR invalid duplication of Object Type line
INVALID_ATTRIBUTE_VALUE invalid duplication of Attribute (Reporting Object Type) line
SYNTAX_ERROR invalid duplication of Object Id line
SYNTAX_ERROR invalid duplication of Scope line
SYNTAX_ERROR invalid duplication of Invoke Id line
INVALID_ATTRIBUTE_VALUE invalid duplication of Attribute (Reporting Object Id) line
INVALID_ATTRIBUTE_VALUE invalid duplication of Attribute (Reporting Object Scope) line
SYNTAX_ERROR unrecognized operator in SET request
(Sheet 5 of 6)

Table 18 (Continued)
Error summary

Error message
INVALID_ATTRIBUTE_NAME invalid attribute in SET request
NOT_SUPPORTED ADD modifier not supported in SET request
NOT_SUPPORTED DELETE modifier not supported in SET request
SYNTAX_ERROR Invalid duplication of Administrative State Attribute
SYNTAX_ERROR Invalid Type for Administrative State Attribute
SYNTAX_ERROR Missing value for Administrative State Attribute
SYNTAX_ERROR Invalid format for Administrative State Attribute value
SYNTAX_ERROR Invalid duplication of Annotation Attribute
SYNTAX_ERROR Invalid Type for Annotation Attribute
SYNTAX_ERROR Invalid format for Annotation Attribute value
INVALID_ATTRIBUTE_NAME Invalid attribute in SET request
SYNTAX_ERROR unrecognized operator in SET request
INVALID_ATTRIBUTE_NAME invalid attribute in SET request
(Sheet 6 of 6)

Index

G

H

I

L

M

N

O

P

R

S

T

V

Preside Multiservice Data Manager

Network Model API

Reference Guide

R15.1

Publication: 241-6001-201
Document status: Standard
Document version: 15.1RSUP
Document date: August 2004
Printed in Canada

NORTEL NETWORKS