Preside Multiservice Data Manager

Provisioning Command Filter API for DPN

Reference Guide

241-6001-209

Preside Multiservice Data Manager

Provisioning Command Filter API for DPN

Reference Guide

Publication: 241-6001-209
Document status: Standard
Document version: 15.1RSUP
Document date: August 2004

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 customers who use the Provisioning Command Filter Application Programming Interface (API) to write custom applications for the Preside Multiservice Data Manager (MDM) workstation.

What you need to know

To use the Provisioning Command Filter API, you need to know how to log on to the Preside Multiservice Data Manager (MDM) workstation. You must be familiar with the UNIX operating system, the UNIX C-shell, and a UNIX text editor such as vi.

Required reading consists of the 241-6001-200 *Preside MDM Application Programming Interface Primer* for an overview of the Application Programming Interface (API). The Provisioning Command Filter API follows the API Primer with the constraints listed in “Compliance statement” (page 89).

You should be familiar with the Component Provisioning tool and the Provisioning API. The Component Provisioning tool is described in 241-6001-012 *Preside MDM Configuration Management for DPN User Guide*. The Provisioning API is described in 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide* and 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

How this document is organized

241-6001-209 *Preside MDM Provisioning Command Filter API Reference Guide* contains the following sections:

- "Introducing the Provisioning Command Filter" (page 15) describes the managed objects and API messages of the Provisioning Command Filter API.
- "Object model" (page 33) describes the containment hierarchy and the object classes of the Provisioning Command Filter API.
- "Using the Provisioning Command Filter API" (page 35) details how to use the Provisioning Command Filter API through a set of examples.
- "Event notifications" (page 53) discusses the event notifications that are exchanged between the Provisioning Command Filter API Provider and the Provisioning Command Filter API User.
- "Using the Provisioning Command Filter API interactively" (page 59) describes how to use the Provisioning Command Filter API interactively.
- "Customer Application Examples" (page 61) provides examples of customer applications for the Provisioning Command Filter API.
- "DPN PE Image Semantic Check Filter" (page 77) describes an API command filter for validating port service and PE_Loader Load file name changes.
- "Post-Download: a specific filter" (page 81) provides customer examples of API Command Filter applications, such as post-download applications.
- "Provisioning Command Filter API object definitions" (page 87) is a reference to the file containing information on the Provisioning Command Filter API object model.

- “Compliance statement” (page 89) details how the Provisioning Command Filter API complies with the API Primer.
- “Error messages” (page 99) provides a list of error messages for the Provisioning Command Filter API.

What’s new in this document

There are no changes in this document for this release.

Text conventions

This document uses the following text conventions:

- `nonproportional spaced plain type`

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

- **`nonproportional spaced bold type`**

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

- *italics*

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

  Words that appear in italics in text are for naming.

- `[optional_parameter]`

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

- `<general_term>`

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

- UPPERCASE,lowercase

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

- |

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

- 241-6001-012 *Preside MDM Configuration Management for DPN User Guide*
- 241-6001-023 *Preside MDM Configuration Management for Passport User Guide*
- 241-6001-200 *Preside MDM Application Programming Interface Primer*
- 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*
- 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*
- 241-6001-303 *Preside MDM Administrator Guide*
- 241-6001-304 *Preside MDM Configuration Management for DPN Administrator Guide*

Chapter 1
Introducing the Provisioning Command Filter

This section introduces the Preside Multiservice Data Manager (MDM) Provisioning Command Filter API and describes its purpose. This section contains the following information:

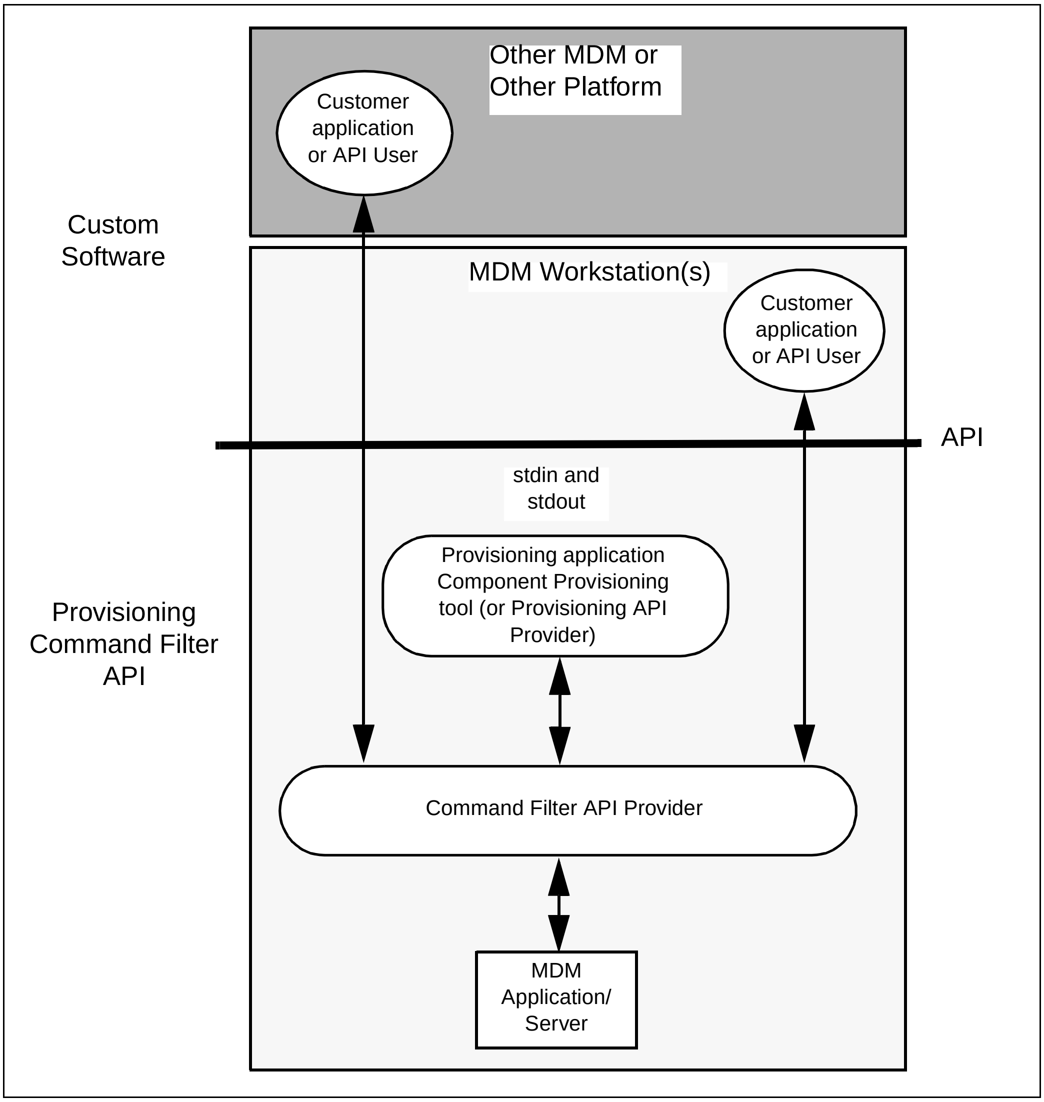
- “What is the Provisioning Command Filter API” (page 15)
- “Provisioning application” (page 17)
- “Using the Provisioning Command Filter API” (page 18)
- “Operational model” (page 19)
- “Object classes” (page 24)

What is the Provisioning Command Filter API

The Provisioning Command Filter API is an ASCII interface that provides a means of controlling which provisioning commands are allowed and which are not. The way that commands are blocked is based on a combination of factors including the command being executed, the specified service data component, and field values.

The Provisioning Command Filter API is one of many Preside Multiservice Data Manager (MDM) APIs. The MDM custom programming environment is shown in the figure “MDM workstation custom programming environment” (page 16). The common behavior and rules for all MDM APIs are described in 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 1
MDM workstation custom programming environment


Other MDM or Other Platform
Customer application or API User
Custom Software
MDM Workstation(s)
Customer application or API User
API
stdin and stdout
Provisioning application Component Provisioning tool (or Provisioning API Provider)
Provisioning Command Filter API
Command Filter API Provider
MDM Application/ Server

Provisioning Command Filter API Provider

The Provisioning Command Filter API Provider is the command interface that provides a means of controlling which provisioning commands are allowed and which are not. The Provisioning Command Filter API Provider is implemented within the provisioning stack between the Field Access server and the provisioning application. The provisioning application can be either the Component Provisioning graphical user interface or the Provisioning API Provider. Each Provisioning Command Filter API Provider supports one provisioning application.

The provisioning stack is fully described in 241-6001-304 *Preside MDM Configuration Management for DPN Administrator Guide*.

Provisioning Command Filter API User

The Provisioning Command Filter API User is the script, program, or human user that communicates with the Provisioning Command Filter API Provider over the interface. The Provisioning Command Filter API User is also referred to as the customer application. The customer application may exist on the Preside Multiservice Data Manager (MDM) platform or on some other platform.

Provisioning application

The provisioning application initiates the requests for service data modifications. The provisioning application can be either the Component Provisioning tool or the Provisioning APIs.

Component Provisioning tool

The Component Provisioning tool is a graphical user interface used to provision DPN-100 modules. For details on DPN Component Provisioning, see 241-6001-012 *Preside MDM Configuration Management for DPN User Guide*.

Provisioning APIs

The DPN and Passport Provisioning APIs are ASCII interfaces that provide access to component provisioning information. Customer-written applications are able to access this component provisioning information through the Provisioning APIs. With the Provisioning APIs you can create, view, and modify service data.

DPN Provisioning API

The DPN Provisioning API Provider is the command interface that allows the DPN Provisioning API User to access the retrieved provisioning service data.

The DPN Provisioning API User is the script, program, or human user that communicates with the DPN Provisioning API Provider over the interface.

For more information on the DPN Provisioning API, see 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*.

Passport Provisioning API

The Passport Provisioning API Provider is the command interface that allows the Passport Provisioning API User to access the retrieved provisioning service data.

The Passport Provisioning API User is the script, program, or human user that communicates with the Passport Provisioning API Provider over the interface.

For more information on the Passport Provisioning API, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

Using the Provisioning Command Filter API

The Provisioning Command Filter API can be used for a variety of customer applications, such as:

- preventing the alteration of selected fields or restricting the values to which they can be set
- restricting the alteration of selected service data to particular users (for example, one group of clerks can only change line data; another group can only change network data)
- displaying confirmation dialogs for delete commands

- populating the Network Reporting System (NRS) after a download
- populating the Network Configuration Database (NCD) after a download
- automatically propagating changes to a downstream MCF
- integrating customer data entry
- automating global data management

You can use various scripting languages to write customer applications for the Provisioning Command Filter API. These include the Bourne shell and the C language.

Some examples are included. For more information on various customer applications for this tool, see “Customer Application Examples” (page 61).

Operational model

This section discusses the operational model of the Provisioning Command Filter API.

Sequence of events

The following sequence of events occur during a typical session with the Provisioning Command Filter API.

1) Provisioning Command Filter API is initialized

The Provisioning Command Filter API is invoked as part of the invocation of the provisioning application. The following are specified on the provisioning invocation command line:

- Component Provisioning tool or Provisioning API parameters,
- Provisioning Command Filter API parameters:
  - — UNIX commands that invoke customer applications, and
  - — nodes and ports of customer application servers.

When the Provisioning Command Filter API Provider is ready to initialize, it sends a version message to each customer application (Provisioning Command Filter API User) that was specified on the command line.

2) Sieves are created and enabled

The customer application then creates and enables the initial set of sieves for the provisioning commands and responses that are to be intercepted.

3) Provisioning commands are sent from the provisioning application

Next, provisioning commands are sent from the provisioning application to the Provisioning Command Filter API Provider. The Provisioning Command Filter API Provider retains commands trapped by sieves until a response is received from the customer application.

4) Provisioning commands are matched to sieves

Commands that are matched to sieves on the Provisioning Command Filter API Provider are sent to the customer application.

5) Commands are either blocked or accepted

The customer application evaluates each command and either accepts or rejects it. The customer application responds with zero or more error messages and an end-of-response. One or more error messages block the command while an end-of-response, by itself, allows the command to be completed.

Before sending a response, the customer application may issue GET requests in order to access additional information. This additional information on service data can help the customer application to decide whether or not the command is valid. For example, it may be important for a customer to maintain certain relationships between fields. Certain fields may always have to be set to the same number. When a change is made to a field, it may be necessary to extract the other field to see if that relationship is being violated or not.

Sieves may be created, altered, or deleted at this time as well. For example, once an MCF has been opened, a set of sieves could be defined that depend on the particular MCF or PM that was opened.

In addition, certain responses can be caught but cannot be blocked. These are the upload and download responses. Although responses cannot be blocked, information and warning messages may be added to the response.

6a) Accepted commands are sent to Field Access

Commands that are accepted are sent from the Provisioning Command Filter API Provider to the Field Access server where the command is completed.

6b) Rejected commands are blocked

If a command is rejected, the customer application sends an error message to the provisioning application where the message is displayed.

7) Provisioning session ends

When the provisioning session ends, the Provisioning Command Filter API sends an _end: message to the customer application.

Command events

When a command is encountered that matches the criteria in one of the sieves, it is formatted into an API request and sent to the customer application as an event notification.

Example 1

The following is an example of a DPN provisioning command that is sent to the customer application as an event notification.

```
_cmd: SET
_obj_class: x25dnacug
_obj_id: compId NI pe 6 pi 6 po 1 x25 0 x25mdna\
x30214034406100 x25dnacug x30214034406100
_mod: REP accountingclass I 23
<blank line>
```

The customer application must then return either an empty response to allow completion of the command or an error response to disallow the completion of the command.

Example 2

The following is an example of an empty response.

```
_end_resp:
<blank line>
```

Example 3

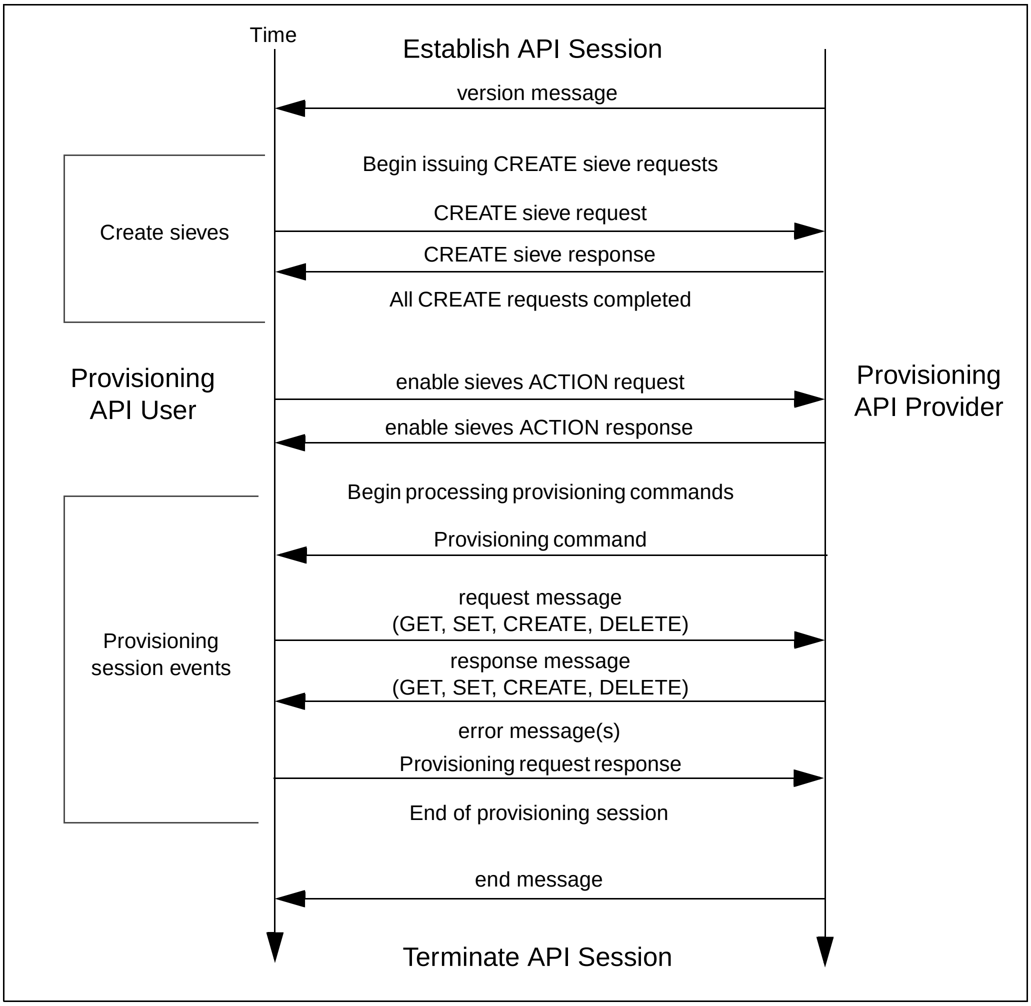
The following is an example of an error response.

```
_error: APPLICATION_ERROR
_attr: errorSeverity S "Error"
_attr: errorCode S "C0001"
_attr: errorApplicationId S "Customer"
_attr: errorFacilityId S "Range"
_attr: errorInformation FS "Only Accounting Class 1 is\
permitted on X.25 lines"
<blank line>
_end_resp:
<blank line>
```

Time line of session events

The figure “Time line diagram” (page 23) shows an overview of the session events. It represents the normal sequence of messages between the Provisioning Command Filter API Provider and the API User. For more details on message types, see 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 2
Time line diagram


Time
Establish API Session
version message
Begin issuing CREATE sieve requests
Create sieves
CREATE sieve request
CREATE sieve response
All CREATE requests completed
Provisioning API User
enable sieves ACTION request
enable sieves ACTION response
Provisioning API Provider
Begin processing provisioning commands
Provisioning command
request message (GET, SET, CREATE, DELETE)
Provisioning session events
response message (GET, SET, CREATE, DELETE)
error message(s)
Provisioning request response
End of provisioning session
end message
Terminate API Session

Operational considerations

Service data cannot be altered by sending requests to the Provisioning Command Filter API. Also, requests and responses cannot be sent to the Provisioning Command Filter API asynchronously.

Requests and responses that are exchanged between the Provisioning Command Filter API and the customer application are in Provisioning API format.

When blocking a command, error records can specify an errorSeverity of Error or Fatal_Error only. Once a customer application has blocked a command, any customer applications that have not yet been contacted are not contacted. That is, the first customer application to block a command stops the process of consulting customer applications for that particular command.

When adding messages to a response, error records can specify an errorSeverity of Information or Warning only. All customer applications that have a sieve that catches the response, will see the response and will be able to add messages to the response. The process does not stop at the first customer application that adds some messages to the response.

Object classes

With the addition of sieves, the Provisioning Command Filter API supports all of the object classes that are supported by the Passport and DPN Provisioning APIs.

For DPN, an online file called /opt/MagellanNMS/lib/api/provapi.model is provided. It contains the object and attribute names as they appear in the DPN Provisioning API model. For format descriptions of DPN object classes, see 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*.

For Passport, a provisioning API model generator allows you to generate object class, attribute, or hierarchy files. By using the query option, you can find out information about attribute or object classes. For details, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

For format descriptions of Passport object classes, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

Data types

The Provisioning Command Filter API supports the following data types:

- I (integer)
- NI (node identifier)

- S (string)
- SS (sequence of strings)
- FS (formatted string)
- H (hexadecimal)
- B (boolean)
- SI (sequence of integers)
- RI (range of integers)
- RS (range of strings)
- SB (sequence of booleans)
- L (long integer)

Sieves and event report types

The sieve objects of the Provisioning Command Filter API control the flow of notifications to the API User.

The standard API event report types are not implemented in the Provisioning Command Filter API. Instead, the Provisioning Command Filter API uses Provisioning API commands and in some cases responses to those commands, as event notifications. These event notifications are used by the Provisioning Command Filter API User to decide whether to block a command or allow it to run to completion.

Message types

The following types of messages are exchanged between the Provisioning Command Filter API Provider and the Provisioning Command Filter API User (customer application):

- version message
- request message
- end message
- response message
- error message
- end-of-response message

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

The Provisioning Command Filter API Provider returns an error response if there is a failure in the processing of a request. After sending an error response, the Provisioning Command Filter API Provider stops processing the request.

The end-of-response message indicates that no more responses are forthcoming and that processing of the request is complete.

In addition, once an enable sieves ACTION request has been successfully completed, the Provisioning Command Filter API Provider presents the API User with Provisioning API commands and certain responses (depending on the sieves that have been defined and what is happening in the Provisioning Application). The API User is expected to respond with zero or more error messages, followed by an end-of-response message to pass or fail the Provisioning request. Before responding, the API user may issue GET requests or manipulate the Provisioning Command Filter sieves using GET, SET, CREATE, and/or DELETE requests.

Request message format

The general format of the request message is:

```
_cmd: <request keyword>
[_inv_id: <invoke ID>]
<request data line>
<request data line>
...
<request data line>
<blank line>

OR

_cmd: ACTION
[_inv_id: <invoke ID>]
_action_type: <action keyword>
<request data line>
<request data line>
```

```
...
<request data line>
<blank line>
```

where:

- `<request keyword>`
  is one of: REGISTER, GET, SET, CREATE, or DELETE.
- `<invoke ID>`
  is an optional integer number supplied by the customer application. The <invoke ID> is returned on responses.
- `<action keyword>`
  is one of: UPLOAD, DOWNLOAD, QUIT, DISCARD, VERIFY, or ENABLE_SIEVES.
- `<request data line>`
  contains data that is specific to the request.

Response message format

The general format of the response message is:

```
[_inv_id: <invoke ID>]
<response data line>
<response data line>
...
<response data line>
<blank line>
```

where:

- `<invoke ID>`
  contains the value that was supplied on the _cmd: line.
- `<response data line>`
  contains data that is specific to the response. If the request was successful, then this contains the data that resulted. If the request failed, then this contains the text of the error message.

A response consists of zero or more response messages terminated by an end-of-response message.

Error message format

The format of an error message is:

```
_error: <API error code>
[_inv_id: <invoke ID>]
_attr: errorSeverity E <severity>
_attr: errorCode S <appl error code>
_attr: errorApplicationId S <appl ID>
_attr: errorFacility S <facility>
_attr: errorText FS <textual error message>
_attr: errorInformation SS <extra data value>
_attr: errorInformation SS <extra data value>
...
_attr: errorInformation SS <extra data value>
<blank line>
```

where:

- `<invoke ID>`
  is the value that was supplied during the request.

  Note: This is not present when it is the API user who is generating the error message.

- `<severity>`
  is the severity of the error. The severity is one of: Information, Warning, Error, or Fatal_Error.

- `<appl error code>`
  is the textual error code.

  In the case of a customer application generated error message, this is normally C0001 and there is one _attr: errorInformation line containing the full text of the error message.

- `<appl ID>`
  is the application identifier. This specifies the provisioning stack process that generated this error.

- `<facility>`
  is the facility code. This specifies which facility within the application layer of the provisioning stack was responsible for generating this error.

- `<textual error message>`
  is the formatted text of the error message.

  Note: This line is not supplied when it is the API user who is generating the error message.

- `<extra data value>`
  is additional data associated with the error message.

  Note: In the case of an API user-generated error message using the C0001 error message code, this is the full formatted text of the error message.

More than one error message can be returned in response to a request. An end-of-response message is used to mark the end of an error response.

Example

```
_error: APPLICATION_ERROR
_attr: errorSeverity S "Error"
_attr: errorCode S "S0101"
_attr: errorApplicationId S "PRO_FA"
_attr: errorFacilityId S "SEMANTIC"
_attr: errorText FS "PE 7 PI 7 PO 5 ITI 0 ITILINK 0:\
\"In Speed\" \ not equal to \"Out Speed\" when speed\
is 110 or AUTO"
_attr: errorInformation SS PE 7 PI 7 PO 5 ITI 0 ITILINK\
0

_end_resp: ACTION
_time: 1995 04 03 17 14 18
```

Note: The Error_Text attribute is not present in error messages sent by the customer application to the Provisioning Command Filter API Provider.

For a list of Provisioning Command Filter API error codes and their description, see “Error messages” (page 99).

End-of-response format

The format of the end-of-response message is:

```
[_inv_id: <invoke ID>]
_end_resp: <request>
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Note: The _time: message is not supplied on response messages generated by the customer application.

For details on the request, response, and error message formats, see 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Supported request messages

For the Provisioning Command Filter API, the supported request messages are:

- REGISTER
  The REGISTER message that is sent by the provisioning application is received by the Provisioning Command Filter API User as an event.
- CREATE
  The CREATE message is supported for sieves and as an event for service data.
- DELETE
  The DELETE message is supported for sieves and as an event for service data.
- GET
  The GET message is supported for sieves, or the uploaded object and all of its subordinates. It is also supported as an event for service data.
- SET
  The SET message is supported for sieves and as an event for service data.
- ACTION
  - enable_sieves
    The enable_sieves ACTION is supported for sieves only.
  - upload
    The upload ACTION is supported as an event for service data.

- download
  The download ACTION is supported as an event for service data.
- verify
  The verify ACTION is supported as an event for service data.
- discard
  The discard ACTION is supported as an event for service data.
- quit
  The quit ACTION is supported as an event for service data.

Supported response messages

The Provisioning Command Filter API accepts the following responses:

- empty or null
- error

When the customer application chooses to block a command, _error: message severities of Error and Fatal_Error are supported.

Chapter 2 Object model

With the addition of sieves, the Provisioning Command Filter API uses the same object model that is used by the Provisioning API. This chapter contains the following:

- "Sieve containment hierarchy" (page 33)
- "Object classes" (page 34)
- "Attributes" (page 34)

For details on the DPN Provisioning API object model, see 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*.

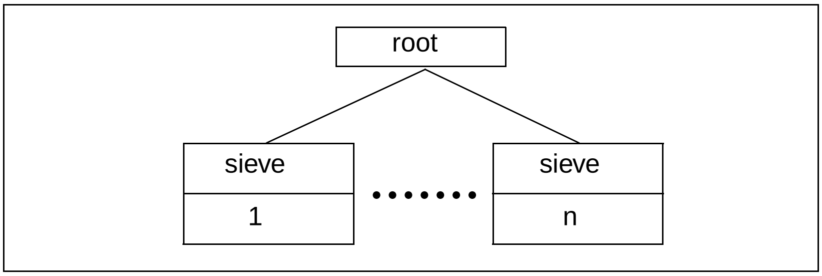
For details on the Passport Provisioning API object model, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

Sieve containment hierarchy

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 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. The sieve containment hierarchy is illustrated in the figure "Sieve containment hierarchy" (page 34).

Figure 3
Sieve containment hierarchy



Since only base scoping is allowed (see “Compliance statement” (page 89)), sieve objects can only be addressed using their names.

Each customer application gets its own set of sieves and cannot see sieves that belong to another customer application.

Object classes

The Provisioning Command Filter API supports all Passport and DPN Provisioning API object classes with the addition of the sieve object class.

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

Attributes

The Provisioning Command Filter API supports the eventFilter attribute and the admState attribute.

Chapter 3
Using the Provisioning Command Filter API

This section explains how to use the Provisioning Command Filter API. This section contains the following information:

- “Code conventions” (page 35)
- “Installing the Provisioning Command Filter API” (page 35)
- “Configuring the Provisioning Command Filter API” (page 35)
- “Terminating Provisioning Command Filter API access” (page 39)
- “Commands” (page 41)

Code conventions

There are two code conventions used in this document.

- A backslash (\ ) at the end of a line indicates that the line of code is continued on the next line.
- A message line that starts with an asterisk (*) or an octothorpe (#) is treated as a comment.

Installing the Provisioning Command Filter API

No further steps are required to install the Provisioning Command Filter API.

Configuring the Provisioning Command Filter API

The Provisioning Command Filter API is not started or stopped directly. It starts indirectly when the provisioning application starts. The DPN Component Provisioning tool and the DPN Provisioning API start the Provisioning Command Filter API

Command Filter parameters

The following options are used to invoke the Provisioning Command Filter API indirectly through the pui or provapi command.

```
<provisioning command>
   [-filter_appl "<UNIX command>" |
   -filter_serv <node/IP address> <port>]
   [-filter_trace] [-filter_width <width>]
   [-falcon]
```

where:

- `<provisioning command>`
  Either /opt/MagellanNMS/bin/pui (with or without the -falcon option), or /opt/MagellanNMS/bin/provapi, or /opt/MagellanNMS/bin/pprovapi. This will usually be the pui command but will also work with the provapi/pprovapi commands.

  If the Provisioning Command Filter API is being invoked through the Component Provisioning graphical user interface, then you will have to modify the /opt/MagellanNMS/bin/pui command with the appropriate Provisioning Command Filter API options for command line invocation.

  If the Provisioning Command Filter API is being invoked through the Provisioning API, then you will have to modify the /opt/MagellanNMS/bin/provapi command with the appropriate Provisioning Command Filter API options.

  To activate command filters permanently, you can add them to the configuration file. See “Configuring for DPN Component Provisioning invocation” (page 38). .

- `-filter_appl`
  An option used to specify a UNIX command that is to be the customer application for command filtering. The Provisioning Command Filter API Provider sets up stdin and stdout for the customer application.

- `<UNIX command>`
  The UNIX Bourne shell command for invoking the customer application that is to filter the provisioning commands.

- `-filter_serv`
  An option used to specify a node name or IP address and port of a server that is to be the customer application for command filtering. The Provisioning Command Filter API Provider makes a stream socket call to the server.
- `<node/IP address>`
  Either the mnemonic or the IP address for a workstation running a Provisioning Command Filter server.
- `<port>`
  The port of the Provisioning Command Filter server.
- `-filter_trace`
  Turns on Provisioning Command Filter API tracing. Trace output is to stderr.
- `-filter_width`
  An option used to specify the maximum length of lines output by the Provisioning Command Filter API Provider. The default is 72.
- `<width>`
  A number between 72 and 100000.
- `-falcon`
  (For Passport only.) An option used to indicate a Passport provisioning session.

  Note: The -falcon option is used only with the pui command.

If initialization fails, the provisioning session is aborted.

The Provisioning Command Filter API remains active for the duration of the provisioning session.

More than one customer application can be started by repeating one of the above options. Sieves are bound to the applications that created them. Commands are presented to customer applications in sequence. The command can only be completed if all customer applications allow it through the filter.

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

```
_version: x.y DPN Provisioning Command Filter API\
(c) 1999 Nortel Networks
```

or

```
_version: x.y Passport Provisioning Command Filter\
API (c) 1999 Nortel Networks
```

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

Configuring for DPN Component Provisioning invocation

You use the configuration file for the Provisioning Command Filter API to specify the filter or filters that you want used for all DPN Component Provisioning sessions. The configuration file is called PUIDpnCmd.cfg and is located in /opt/MagellanNMS/cfg. You can modify the configuration file and add one or more filters.

All future DPN Component Provisioning sessions will use the filter or filters. You can start sessions from the Preside MDM window or from other tools such as Network Viewer (NV) and Component Information Viewer (CIV).

For example, you can add the following to the configuration file to activate the population of Network Reporting System after a successful download of the MCF:

```
-filter_appl "/opt/MagellanNMS/bin/acf_post_download
        /opt/MagellanNMS/bin/acf_pd_nrspop"
-filter_width 100000
```

Configuring for DPN Provisioning API invocation

If you are using the DPN Provisioning API to modify service data, you need to modify the invocation command for the DPN Provisioning API (/opt/MagellanNMS/bin/provapi) to include the options for the Provisioning Command Filter API. Once this is done, the Provisioning Command Filter API is invoked when the DPN Provisioning API is invoked (through the /opt/MagellanNMS/bin/provapi command).

Nortel Networks supplies the following scripts that invoke the DPN Provisioning API. You need to alter them.

/opt/MagellanNMS/lib/macros/nms/dbnl_passwd_if_api
/opt/MagellanNMS/bin/ntippistart
/opt/MagellanNMS/bin/swsppistart
/opt/MagellanNMS/bin/propagate
/opt/MagellanNMS/bin/register

Customer applications are invoked or contacted by the Provisioning Command Filter API.

Configuring for Passport Provisioning API invocation

If you are using the Passport Provisioning API to modify service data, you need to modify the invocation command for the Passport Provisioning API (/opt/MagellanNMS/bin/pprovapi) to include the options for the Provisioning Command Filter API. Once this is done, the Provisioning Command Filter API is invoked when the Passport Provisioning API is invoked (through the /opt/MagellanNMS/bin/pprovapi command).

Nortel Networks supplies the following scripts that invoke the Passport Provisioning API. You need to alter them.

/opt/MagellanNMS/bin/ppropagate
/opt/MagellanNMS/bin/pregister

Customer applications are invoked or contacted by the Provisioning Command Filter API.

Terminating Provisioning Command Filter API access

The Provisioning Command Filter API is normally not terminated from a customer application. It is terminated when the provisioning application terminates. In exceptional situations, the provisioning session can be terminated by sending an _end: message from a customer application.

If the Provisioning Command Filter API receives a premature EOF from a customer application, all commands trapped by the sieves for that customer application are failed with an error message stating that the customer application terminated prematurely.

To terminate the Provisioning session, use the following message:

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

The _end: message is sent when the provisioning session has ended. Normally the Provisioning Command Filter API sends this request to the customer application when the provisioning session ends.

The message is:

```
_end: <termination code>
<blank line>
```

where `<termination code>` can be one of the following:

- USER_REQUEST
  The customer application has requested termination with the _end: request.
- FATAL_ERROR
  One of the following unrecoverable errors occurred:
  - PDU failure
    The Provisioning Command Filter API was unable to send or receive a Protocol Data Unit (PDU). A PDU is a specially formatted IPC message exchanged by provisioning applications. A PDU failure might indicate a software problem, or in the case of a large PDU, it might mean that the /tmp directory has run out of space.
  - Out of memory
    An attempt to allocate some memory has failed.
- LOST_CONNECTION
  There is a software problem, or another type of problem, in the provisioning application, Field Access server, Envelope Access server, File Access server, or in the IPC.

Commands

Each command has a request message and a response, and/or error message(s). See 241-6001-200 *Preside MDM Application Programming Interface Primer*, for more details on commands. The Provisioning Command Filter API supports the following commands:

- CREATE
- GET
- SET
- DELETE
- ACTION
  - enable_sieves

The CREATE command

Use the CREATE command to create new sieves. Creation of service data is not allowed.

Using the CREATE command

The format of the CREATE request message is:

```
_cmd: CREATE
[_inv_id: <invoke ID>]
_obj_class: sieve
[_attr: admState I <state>]
_attr: eventFilter SS <attr name> <op> <attr type &
value>
...
_attr: eventFilter SS <attr name> <op> <attr type &
value>
<blank line>
```

where:

- `<state>`
  0 (locked) or 1 (unlocked)
  Default is 1 (unlocked).
- `<op>`
  One of EQ, NE, GT, LT, GE, LE, or P.

- `<attr name>`
  Can be the name of any attribute or component category name found in the /opt/MagellanNMS/lib/api/provapi.model file or the Passport Provisioning API model file. For Passport Provisioning, the ID or ABBREVIATION can also be used. Can also be command, action, or response.

  You can qualify attribute or category names by specifying parent category names in their proper hierarchical order. For example, em/nmis/local/Session. Not all parent names need to be specified, only what is needed to get the desired result.

- `<attr type & value>`
  Possible values are:

  — If `<attr name>` is command, then `<attr type & value>` is one of: S GET, S SET, S CREATE, S DELETE, S REGISTER, S ACTION, or nothing (if the P operator is used).

  — If `<attr name>` is action, then `<attr type & value>` is one of: S upload, S download, S discard, S verify, S quit, or nothing (if the P operator is used).

  — If `<attr name>` is response, then `<attr type & value>` is S ACTION or nothing (if the P operator is used). Only upload and download ACTION responses are trapped.

  — If `<attr name>` is the name of a service data field then `<attr type & value>` is a valid type and value for that field.

  — If `<attr name>` is a component category name, the attribute value type is S and the attribute value is a string key value.

The format of the CREATE response message is:

```
[_inv_id: <invoke ID>]
_obj_class: sieve
_obj_id: sieveId I <integer>
<blank line>
_end_resp:
[_inv_id: <invoke ID>]
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

The result of filter elements with the same <attr name> are ORed. Those results are then ANDed.

Example
The following is an example of a CREATE request:

```
_cmd: CREATE
_obj_class: sieve
_attr: eventFilter SS command eq S "SET"
_attr: eventFilter SS x25dnacug P
_attr: eventFilter SS accountingclass P
```

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

```
_obj_class: sieve
_obj_id: sieveId I 123
<blank line>
_end_resp:
_time: 1993 08 19 17 14 18
<blank line>
```

Any time the accounting class field in an X25 DNA CUG envelope is set, the API request for that change is sent to the customer application. The customer application must then reply with either an empty response or an error response. The customer application must decide whether the new accounting class value is acceptable or not and return a response that reflects that decision. An empty response causes the change request to be completed. An error response aborts the change request and the supplied error message is displayed. More than one error message may be supplied and displayed.

The GET command

The GET command retrieves service data or information about sieves. The amount of service data returned can be tailored using the _scope: line.

The _attr_id: line is used to select the attribute(s) to be returned. If ALL is specified as an attribute name, then all attributes are returned. If no _attr_id: line is coded, then only the names of components are returned. More than one _attr_id: line may be coded.

The options for the _scope: line are:

- BASE
  returns the attributes for the component named by the _obj_id: line. This is the default value if the _scope: line is not used.
- NEXT
  returns the names of the objects at the next lower level and their attributes. A separate response message is returned for each object.
- ALL
  returns the names of the objects for the entire subhierarchy and their attributes. A separate response message is returned for each object.

Using the GET command

The format of the GET request message is:

```
_cmd: GET
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <distinguished name>
[_scope: BASE | NEXT | ALL]
[_attr_id: <attribute name> | ALL]
<blank line>
```

The format of the GET response message is:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
_attr: <attribute name> <attribute type> <attribute
value>
_attr: <attribute name> <attribute type> <attribute
value>
...
_attr: <attribute name> <attribute type> <attribute
value>
<blank line>
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
_attr: <attribute name> <attribute type> <attribute
value>
_attr: <attribute name> <attribute type> <attribute
```

```
value>
...
_attr: <attribute name> <attribute type> <attribute
value>
<blank line>
...
...
...
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
_attr: <attribute name> <attribute type> <attribute
value>
_attr: <attribute name> <attribute type> <attribute
value>
...
_attr: <attribute name> <attribute type> <attribute
value>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: GET
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

To obtain all the service data for PE 5 PI 5 PO 1 ITI 0 and itilink 0, issue the following request:

```
_cmd: GET
_obj_id: compId NI pe 5 pi 5 po 1 iti 0 itilink 0
_attr_id: ALL
<blank line>
```

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

```
_obj_class: itilink
_obj_id: compId NI PE 5 PI 5 PO 1 ITI 0 ITILINK 0
_attr: parity S “Auto”
_attr: characterlength S “7 bits”
_attr: inputspeed I 1200
_attr: outputspeed I 1200
_attr: modem I 113
```

```
_attr: stopbitclass S “1 over 110 bpc”
_attr: portmode S “DTE”
_attr: busyoutallowed I 0
_attr: callretries S “Unlimited”
_attr: forwardeverycharacter I 0
_attr: idletimerinterval S “50 msec”
_attr: cts I 0
_attr: rts I 0
_attr: stimer S “Infinity”
_attr: rtimer S “Infinity”
_attr: maximumassemblytimer S “Infinity”
_attr: callsetuptimer I 1
<blank line>
_end_resp: GET
_time: 1993 08 19 17 14 18
<blank line>
```

Note: For Passport, you can specify indexed attributes but without any indexes. All indexed values are returned.

GET requests are also valid for existing sieves. The object class is sieve and the object id has the following form:

```
_obj_id: sieveId I <sieveid>
```

Only BASE scope is supported for sieve GET requests.

Example

To obtain all the data for sieve 1, enter the following request:

```
_cmd: GET
_obj_class: sieve
_obj_id: sieveID I 1
_attr_id: ALL
<blank line>
```

If the GET request is processed successfully, the response will be similar to:

```
_obj_class: sieve
_obj_id: sieveId I 1
_attr: admState I 1
```

```
_attr: eventFilter SS command P
<blank line>
_end_resp: GET
_time: 1995 10 30 11 14 37
<blank line>
```

The SET command

The SET command modifies the specified attribute values of the specified sieve. Service data cannot be altered.

Note: Only BASE scope is supported. Filtering is not supported.

Using the SET command

The format of the SET request message is:

```
_cmd: SET
[_inv_id: <invoke ID>]
_obj_class: sieve
_obj_id: sieveId I <sieve id>
[_scope: BASE]
_mod: <op> admState I <state>
_mod: <mod> eventFilter SS <attr name> <op> <attr type>
<attr value>
_mod: <mod> eventFilter SS <attr name> <op> <attr type>
<attr value>
...
_mod: <mod> eventFilter SS <attr name> <op> <attr type>
<attr value>
<blank line>
```

where:

- `<mod>`
  This is one of: ADD, DEL, REP, or DEF:
  - ADD
    Add the specified test to the sieve. No error is indicated if the test already exists.
  - DEL
    Delete the specified test from the sieve. No error is indicated if the test does not exist.

— REP
  All existing tests referring to the same <attr name> are deleted. Then the test is added to the sieve. For the admState attribute, its value is set.

— DEF
  Reset the attribute to its default. Only the admState attribute has a default value. It is 1.

The other items are the same as in the CREATE request.

The format of the SET response message is:

```
[_inv_id: <invoke ID>]
_obj_class: sieve
_obj_id: sieveId I <sieve id>
<blank line>
_end_resp:
[_inv_id: <invoke ID>]
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example
To set the sieve to match attributes with an input speed of 2400, issue the following request:

```
_cmd: SET
_obj_class: sieve
_obj_id: sieveId I 456
_attr: eventFilter SS inputspeed eq I 2400
<blank line>
```

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

```
_obj_class: sieve
_obj_id: sieveId I 456
<blank line>
_end_resp:
_time: 1995 03 19 17 14 18
<blank line>
```

Anytime the input speed field in an ITILINK envelope is changed to 2400, the API request for that change is sent to the customer application.

The DELETE command

The DELETE command deletes the indicated sieve. Service data cannot be deleted.

Using the DELETE command

The format of the DELETE request message is:

```
_cmd: DELETE
[_inv_id: <invoke ID>]
_obj_class: sieve
_obj_id: sieveId I <sieve id>
<blank line>
```

The format of the DELETE response message is:

```
[_inv_id: <invoke ID>]
_obj_class: sieve
_obj_id: sieveId I <sieve id>
<blank line>
_end_resp:
[_inv_id: <invoke ID>]
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

To delete a sieve with a component identifier of 456, issue the following request:

```
_cmd: DELETE
_obj_class: sieve
_obj_id: sieveId I 456
<blank line>
```

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

```
_obj_class: sieve
_obj_id: sieveId I 456
<blank line>
```

```
_end_resp:
_time: 1995 03 19 17 14 18
<blank line>
```

ACTION

The ACTION request identifies one object that is to perform a defined action. An ACTION request results in one successful ACTION response message with zero or more Information or Warning error messages, or error message(s) followed by an end-of-response message.

The enable_sieves ACTION

The enable_sieves action enables the sieves to begin processing provisioning commands. The processing of the first provisioning command is delayed until this command has been issued by all customer applications.

The format of the enable_sieves action request message is:

```
_cmd: ACTION
_obj_class: root
_obj_id: rootId NI
[_inv_id: <invoke ID>]
_action_type: Enable_Sieves
<blank line>
```

The format of the enable_sieves action response message is:

```
[_inv_id: <invoke ID>]
_obj_class: root
_obj_id: rootId NI
<blank line>
_end_resp:
[_inv_id: <invoke ID>]
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

A sample request message follows:

```
_cmd: ACTION
_obj_class: root
_obj_id: rootId NI
_action_type: Enable_Sieves
```

A sample response follows:

```
_obj_class: root
_obj_id: rootId NI
<blank line>
_end_resp:
_time: 1995 04 19 17 14 18
<blank line>
```

Chapter 4 Event notifications

This section discusses the event notifications that are exchanged between the Provisioning Command Filter API Provider and the Provisioning Command Filter API User. This section contains the following information:

Event notifications

The event notifications that are exchanged between the Provisioning Command Filter API Provider and the Provisioning Command Filter API User are actually either provisioning commands or in some cases responses to those commands. These event notifications are used by the Provisioning Command Filter API to decide which provisioning commands are accepted and which are not. Accepted commands are allowed to continue on to the Field Access server where they run to completion. Commands that are not accepted are blocked by the Provisioning Command Filter API.

Upload and download action response events may be preceded by warning and/or information messages associated with the response event. Any such warning and/or information messages are not replied by the Customer Application. The actual response event is in the last message presented. It is this message that is replied by the Customer Application.

For DPN, examples of the event notifications as issued by the Provisioning API are found in 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*. For Passport, event notification examples are found in 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

CREATE notification

The format of the CREATE notification is:

```
_cmd: CREATE
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
[_attr: <attribute name> <attribute type>\
<attribute value>]
[_attr: <attribute name> <attribute type>\
<attribute value>]
...
[_attr: <attribute name> <attribute type>\
<attribute value>]
<blank line>
```

GET notification

The format of the GET notification is:

```
_cmd: GET
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
[_scope: BASE | NEXT | ALL]
[_attr_id: <attribute name> | ALL]
[_attr_id: <attribute name> | ALL]
...
[_attr_id: <attribute name> | ALL]
<blank line>
```

SET notification

The format of the SET notification is:

```
_cmd: SET
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
[_scope: BASE]
_mod: REP <attribute name> <attribute type> <attribute
value>
_mod: REP <attribute name> <attribute type> <attribute
value>
...
_mod: REP <attribute name> <attribute type> <attribute
value>
<blank line>
```

DELETE notification

The format of the DELETE notification is:

```
_cmd: DELETE
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
```

REGISTER notification

For DPN, the format of the REGISTER notification is:

```
_cmd: REGISTER
_user_id: <UNIX user ID>
[_attr: destinationmnemonic S <destination mnemonic>]
[_attr: capabilityid S <capability ID]
[_attr: password S <password>]
<blank line>
```

For Passport, the format of the REGISTER notification is:

```
_cmd: REGISTER
[_inv_id: <invoke ID>]
_attr: group S <group name>
_attr: capabilityid S <value>
_attr: password S <password>
<blank line>
```

Upload ACTION notification

For DPN, the format of the upload action notification for creating new service data is:

```
_cmd: ACTION
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
_action_type: upload
_attr: accessmode S UPDATE
_attr: sdversion I <version>
<blank line>
```

For DPN, the format of the upload action notification for existing service data is:

```
_cmd: ACTION
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
_action_type: upload
[_attr: accessmode S READ | UPDATE]
[_attr: location SS PM <PM> | DISK]
_attr: view SS bundle <bundle> | COMMITTED | ACTIVE | \
key <key> | datekey <date key>
[_attr: namsid I <NAMS ID>]
<blank line>
```

For Passport, the format of the upload action notification is:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
_action_type: upload
[_attr: preload S [yes|no]]
[_attr: accessmode S READ | UPDATE]
[_attr: view SS viewname <view filename> | edit |
committed | current | key <key> | datekey <date key>]
<blank line>
```

Download ACTION notification

For DPN, the format of the download action notification is:

```
_cmd: ACTION
_obj_class: <object class>
_obj_id: compId NI PM <PM>
_action_type: download
[_attr: location SS PM <PM> | DISK]
_attr: view SS bundle <bundle> | key <key> | datekey\
<date key>
[_attr: activationdate S <YYMMDD>]
[_attr: mode S COMPLETE | INCREMENTAL]
[_attr: check S COMPLETE | INCREMENTAL | MINIMAL]
_attr: activatable S YES | NO
[_attr: namsid I <NAMS ID>]
<blank line>
```

For Passport, the format of the download action notification is:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
_action_type: download
attr: view SS viewname <view filename> | key <key> |
datekey <date key>
_attr: check S [NOCHECK|CHANGED|COMPLETE]
_attr: stoponerr S [YES|NO]
[_attr: save SS [ASCII] [PORTABLE]]
<blank line>
```

Upload ACTION response notification

For DPN, the format of the upload action response notification is:

```
_obj_class: <object class>
_obj_id: compId NI PM <distinguished name>
[_attr: mcfname S <MCF name>]
[_attr: activationdate S <YYMMDD>]
_attr: sdversion I <SD version>
<blank line>
```

For Passport, the format of the upload action response notification is:

```
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
[_attr: view S <view name>]
_attr: sdversion S <SD version>
_attr: swversion S <Passport version>
<blank line>
```

Download ACTION response notification

For DPN, the format of the download action response notification is:

```
_obj_class: <object class>
_obj_id: compId NI PM <PM>
_attr: mcfname S <MCF name>
<blank line>
```

For Passport, the format of the download action response notification is:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI EM <Passport name>
_attr: viewname S <view name>
<blank line>
```

Chapter 5
Using the Provisioning Command Filter API interactively

This section describes how to use the Provisioning Command Filter API interactively. This sections contains the following information:

- "Learning to use the API interactively" (page 59)

Learning to use the API interactively

The easiest way to learn to use the Provisioning Command Filter API is interactively. The C source files for two interactive interfaces are provided. You can compile and link them using the following UNIX commands:

```
cc -O /opt/MagellanNMS/cfg/macros/nms/src/ttyappl.c \
      -o ttyappl
cc -O /opt/MagellanNMS/cfg/macros/nms/src/ttyserv.c \
      -o ttyserv -lsocket
```

The tty appl application is invoked when you specify a -filter_appl ttyappl option on a pui or provapi command line. The ttyappl application directs output from the Provisioning Command Filter API Provider to the window from which you invoked pui or provapi. Anything that you input in that window is sent to the Provisioning Command Filter API Provider.

The ttyserv server is started when you enter a ttyserv <port> UNIX command. The <port> is any unused stream socket port that is not reserved. You can try numbers starting at 4000 and up. To have the Provisioning Command Filter API Provider connect to this server, enter a pui or provapi command with a -filter_serv <node/IP address> <port> option from another window. The <port> is the same as that specified in the ttyserv command.

When the connection is made, the ttyserv program displays a message indicating that an incoming call has been accepted.

Once you’ve started pui or provapi, the _version: message appears in the window. There may be a slight delay. At this point, you can interactively use the Provisioning Command Filter API. You can manually enter requests. Leave an edit window open for requests and responses. You can cut and paste individual requests and responses from the edit window to the ttyappl or ttyserv window. You can also make and save corrections in the edit window.

Chapter 6
Customer Application Examples

This section provides examples of customer applications for the Provisioning Command Filter API. This section contains the following information:

The source code for these examples are located in the following directory:

/opt/MagellanNMS/lib/macros/nms/src

Example 1: Restricting X25 accounting class

This customer application prevents users from changing the X25 accounting class to a value other than 1.

This example is intended for use with DPN Provisioning only.

Example script

Create a Bourne shell script called X25accounting. The script contains the following:

```
# !/bin/sh

# *** Example Software ***
# Nortel Networks provides this example program as is,
# without any warranty express or implied, with respect
# to the program, including but not limited to
# warranties of merchantability or fitness for a
# particular purpose. Nortel Networks disclaims any
# liability of any kind for any damages whatsoever
# resulting from the use of this program, including,
# without limitation, incidental, consequential,
# indirect, or special damages of any kind, even if
# Nortel Networks is aware of the possibility of such
# damages.

# Nortel Networks will not provide any technical
# support for this example program.

cat >/dev/null 2>&1 | echo '_cmd: CREATE
_obj_class: sieve
_attr: admState I 1
_attr: eventfilter SS command eq s create
_attr: eventfilter SS command eq s set
_attr: eventfilter SS x25dnacug p
_attr: eventfilter SS accountingclass ne i 1

_cmd: action
_obj_class: root
_obj_id: rootId NI
_action_type: enable_sieves
'

while :
do
echo '_error:
_attr: errorSeverity E Error
_attr: errorCode S "C0001"
_attr: errorApplicationId S "Customer"
_attr: errorInformation FS "X25 Accounting Class \
must always be 1"

_end_resp:
'
done
```

Data written by the Provisioning Command Filter API to the Customer Application is read by the Customer Application from stdin. Data written by the customer application to stdout is read and interpreted by the Provisioning Command Filter API.

In this example, a single sieve is defined to catch commands that try to change the X25 accounting class field to something other than 1. Since this is all the sieve catches, the command events do not need to be analyzed, and they can be discarded. This is what cat >/dev/null 2>&1 does. However, when you first create a script like this, you should direct the data from the Provisioning Command Filter API to a file in order to see any errors. Once you've debugged the script, you can alter it to discard the data.

The sieve looks for the presence of all of the following:

- a command that alters data

  The CREATE and SET commands alter data. Event filters that refer to the same attribute are OR'ed.

- X25 service data is affected

  An event filter that specifies the presence of the x25dnacug component category name ensures that X25 service data is involved. The X25 DNA CUG component was chosen because the X25 accounting class field is inside this component. However, the category names of the parent components, X25MDNA and X25, would also work.

- an accounting class with a value other than 1

  This requires an event filter that looks for the presence of an accounting class field that is not equal to 1.

After the sieve is defined, you need an enable sieves command to tell the Provisioning Command Filter API that the initial set of sieves has been defined and that the provisioning session and command filtering can proceed. The Provisioning Command Filter API will examine all commands from the provisioning client. When it finds a command that satisfies the conditions in the sieve, it presents it to the customer application. In this particular case, the customer application simply discards it since the sieve selects exactly what is wanted. No further analysis is needed.

The Provisioning Command Filter API then reads a response from the customer application. In this example, the script simply writes an _error: record followed by an _end_resp: record in a loop. So, each time the Provisioning Command Filter API encounters a command that tries to change the X25 accounting class field to something other than 1, it reads an error response from the customer application and blocks the command. The provisioning client (for example, the UI) displays the error message in the error response "X25 Accounting Class must always be 1".

Usage

For this customer application, enter the following to invoke the Provisioning Command Filter API with the Component Provisioning UI:

```
/opt/MagellanNMS/bin/pui -filter_appl x25accounting &
```

To invoke Component Provisioning with the Provisioning Command Filter API from a menu, add the following to the pui command line in the toolset files:

```
-filter_appl x25accounting
```

Example 2: Delete confirmation

This customer application presents a confirmation dialog each time a user tries to delete a component.

This example is intended for use with both DPN and Passport Provisioning.

Example script

This example requires two Bourne shell scripts: confirmdel and confirm. The confirmdel script contains the following:

```
# !/bin/sh

# *** Example Software ***

# Nortel Networks provides this example program as is,
# without any warranty express or implied, with respect
# to the program, including but not limited to
# warranties of merchantability or fitness for a
# particular purpose. Nortel Networks disclaims any
# liability of any kind for any damages whatsoever
# resulting from the use of this program, including,
# without limitation, incidental, consequential,
# indirect, or special damages of any kind, even if
# Nortel Networks is aware of the possiblity of such
# damages.

# Nortel Networks will not provide any technical
# support for this example program.

# Note: Requires the presence of the confirm Bourne
# shell script.

echo '_cmd: CREATE
_obj_class: sieve
_attr: eventfilter SS command eq S delete

_cmd: action
_obj_class: root
_obj_id: rootId NI
_action_type: enable_sieves
'

until test "$line" = "_end_resp: ACTION"
    do
    read line
    if test $? -ne 0
    then exit 0
    fi
    done

until test "$line" = ""
    do
    read line
    if test $? -ne 0
    then exit 0
    fi
    done

respfn="/tmp/confirmdel$$"

while :
    do
    read keyword compid ni id
    if test $? -ne 0
    then exit 0
    fi
    if test "$keyword" = "_obj_id:"
    then until test "$line" = ""
        do
        read line
        if test $? -ne 0
        then exit 0
        fi
        done
  xterm -T "Confirm DELETE" -n "Confirm" -sf -fg \
black -bg white -fn 9x15bold -fb 9x15bold -geometry \
100x5+200+500 -e confirm "DELETE" "$id" "$respfn"
     cat "$respfn"
     rm "$respfn"
    fi
    done
```

The confirmdel script interacts with the Provisioning Command Filter API. It invokes the second script, confirm, as part of bringing up an xterm window.

The confirm script contains the following:

```
# !/bin/sh

# *** Example Software ***

# Nortel Networks provides this example program as is,
# without any warranty express or implied, with respect
# to the program, including but not limited to
# warranties of merchantability or fitness for a
# particular purpose. Nortel Networks disclaims any
# liability of any kind for any damages whatsoever
# resulting from the use of this program, including,
# without limitation, incidental, consequential,
# indirect, or special damages of any kind, even if
# Nortel Networks is aware of the possiblity of such
# damages.

# Nortel Networks will not provide any technical
# support for this example program.

cont() {
echo ’_end_resp:
’ >$1
exit 0
}
```

```
echo ’_error:
_attr: errorSeverity E Error
_attr: errorCode S "C0001"
_attr: errorApplicationId S "Customer"
_attr: errorFacility S "’$1’ Confirmation"
_attr: errorInformation FS "’$1’ aborted by user \
request."

_end_resp:
’ >$3

while :
    do
    echo "Do you really want to $1 component $2? \
 (yes/no)"
    read resp rest
    if test $? -ne 0
    then exit 0
    fi
    case $resp in
        "y"   ) cont "$3";;
        "ye"  ) cont "$3";;
        "yes" ) cont "$3";;
        "n"   ) exit 0;;
        "no"  ) exit 0;;
        esac
    done
```

In this example, the confirmdel script creates a sieve to catch all delete commands. This is followed by the enable sieves command.

Unlike the previous example, the data from the Command Filter cannot be discarded. The name of the component being deleted must be displayed in the confirmation dialog window.

Since data from the Provisioning Command Filter API needs to be analyzed, the first task is to skip over the create sieve and enable sieves command responses. To do this, the customer application reads input until it sees the "_end_resp: ACTION" line. This is the first line of the enable sieves command response. The program then reads input until it encounters a blank line, which marks the end of the enable sieves command response.

Further data from the Provisioning Command Filter API consists of the delete commands caught by the sieve. A unique filename is created for the file containing the response from the confirm script. At this point the confirmdel script reads the command events data that is presented by the Provisioning Command Filter API. The only information of interest is the component ID. It therefore looks for the _obj_id: line and then skips the rest of the command by looking for the blank line that marks its end.

To obtain confirmation of the delete command, the confirmdel script brings up an xterm window and uses the -e option to run the confirm script in that window. This second script asks the user to confirm the delete and accepts the response.

The confirm script takes three parameters: the action being confirmed (Delete in this case), the name of the affected component, and the name of a file in which to pass the response. The script asks the user to confirm the action. It accepts a yes or no response. Based on this input, the script writes a Provisioning Command Filter API response into the response file.

The confirmdel script then sends the contents of the response file to the Provisioning Command Filter API.

Usage

For this customer application, to invoke Component Provisioning with the Provisioning Command Filter API from a menu, add the following to the pui command line in the toolset files:

```
-filter_appl confirmdel -filter_width 100000
```

Example 3: Limiting changes to network service data

This customer application restricts service changes to network data (that is, anything other than ports and their sub-components).

This example is intended for use with DPN Provisioning only.

Example script

This example requires two Bourne shell scripts: limit2net and dispmsg. The limit2net script contains the following:

```
# !/bin/sh
# *** Example Software ***

# Nortel Networks provides this example program as is,
# without any warranty express or implied, with respect
# to the program, including but not limited to
# warranties of merchantability or fitness for a
# particular purpose. Nortel Networks disclaims any
# liability of any kind for any damages whatsoever
# resulting from the use of this program, including,
# without limitation, incidental, consequential,
# indirect, or special damages of any kind, even if
# Nortel Networks is aware of the possibility of such
# damages.

# Nortel Networks will not provide any technical
# support for this example program.

# Note: Requires the presence of the dispmsg Bourne
# shell script.

echo '_cmd: CREATE
_obj_class: sieve
_attr: eventfilter SS command eq S create
_attr: eventfilter SS command eq S delete
_attr: eventfilter SS command eq S set
_attr: eventfilter SS po p

_cmd: action
_obj_class: root
_obj_id: rootId NI
_action_type: enable_sieves
'

# Wait for response for enable sieves ACTION command.

until test "$line" = "_end_resp: ACTION"
    do
    read line
    if test $? -ne 0
    then exit 0
    fi
    done

# Wait for end of response for enable sieves ACTION
# command.
```

```
until test "$line" = ""
    do
    read line
    if test $? -ne 0
    then exit 0
    fi
    done

# Scan trapped commands for the object id command and
# the presence of _mod: lines. Empty SETs (no _mod:
# lines) are passed (EDIT Dialog sends an empty SET
# on CANCEL). Non-empty SETs result in an Error
# Message window. When CREATEs and DELETEs are
# failed, PUI displays a window, so another window
# from here is not needed.

while :
    do
    read keyword compid_cmd ni id
    if test $? -ne 0
    then exit 0
    fi
    if test "$keyword" = "_cmd:"
    then cmd="$compid_cmd"
    fi
    if test "$keyword" = "_obj_id:"
    then modline="no"
       until test "$keyword" = ""
           do
           read keyword rest
           if test $? -ne 0
           then exit 0
           fi
           if test "$keyword" = "_mod:"
           then modline="yes"
           fi
           done
         if test "$modline" = "yes" -o "$cmd" != "SET"
         then if test "$cmd" = "SET"
          then xterm -T "Error Message Dialogue" -n \
"Error" -sf -fg black -bg white -fn 9x15bold -fb \
9x15bold -geometry 100x5+200+500 -e dispmsg \
"Alteration of $id is not allowed."
```

```
            fi
            echo '_error:
_attr: errorSeverity E Error
_attr: errorCode S "C0001"
_attr: errorApplicationId S "Customer"
_attr: errorFacility S "Access Control"
_attr: errorInformation FS "Creation/deletion/alter\
ation of this component is not allowed."
'
       fi
       echo '_end_resp:
'
    fi
    done
```

The limit2net script interacts with the Provisioning Command Filter API. It invokes the second script, dispmsg, as part of bringing up an xterm window.

The dispmsg script contains the following:

```
# !/bin/sh

# *** Example Software ***

# Nortel Networks provides this example program as is,
# without any warranty express or implied, with respect
# to the program, including but not limited to
# warranties of merchantability or fitness for a
# particular purpose. Nortel Networks disclaims any
# liability of any kind for any damages whatsoever
# resulting from the use of this program, including,
# without limitation, incidental, consequential,
# indirect, or special damages of any kind, even if
# Nortel Networks is aware of the possiblity of such
# damages.

# Nortel Networks will not provide any technical
# support for this example program.

echo ""
echo "$1"
echo ""
echo "Press RETURN to continue."

read ignore
```

In this example, component provisioning usually displays a separate window to show the error message. In cases where this does not occur, the limit2net script brings up an xterm window. The x-term window then runs the dispmsg script to display the error message. The dispmsg script exits when the user presses Return.

Regardless of whether or not a window is used to display the error message, an error response is sent to the Provisioning Command Filter API to block the command.

Usage

For this customer application, to invoke Component Provisioning with the Provisioning Command Filter API from a menu, add the following to the pui command line in the toolset files:

```
-filter_appl limit2net -filter_width 100000
```

Example 4: Limiting changes to port service data

This customer application restricts service data changes to ports and their sub-components.

This example is intended for use with DPN Provisioning only.

Example script

This example requires a Bourne shell script, dispmsg, and a C program, limit2po. For more information on the dispmsg script, see “Example 3: Limiting changes to network service data” (page 68). The limit2po.c C source file contains the following:

```
/* *** Example Software *** */

/* Nortel Networks provides this example program */
/* as is, without any warranty express or implied, */
/* with respect to the program, including but not */
/* limited to warranties of merchantability or */
/* fitness for a particular purpose. Nortel Networks */
/* disclaims any liability of any kind for any */
/* damages whatsoever resulting from the use of this */
/* program, including, without limitation, */
/* incidental, consequential, indirect, or special */
/* damages of any kind, even if Nortel Networks is */
/* aware of the possiblity of such damages. */

/* Nortel Networks will not provide any technical */
/* support for this example program. */

/* Note: Requires the presence of the dispmsg shell */
/* script. */

# include <stdio.h>

main()
{

char buf[1000], msgcmd[1000], *reply, cmd[100],
modline;

/* Create and enable sieve to catch data altering */
/* commands. */

printf("_cmd: CREATE\n\
_obj_class: sieve\n\
_attr: eventfilter SS command eq S CREATE\n\
_attr: eventfilter SS command eq S DELETE\n\
_attr: eventfilter SS command eq S SET\n\
\n\
_cmd: action\n\
_obj_class: root\n\
_obj_id: rootId NI\n\
_action_type: enable_sieves\n\
\n");

fflush(stdout);

/* Look for end of response for enable sieves action.*/

do
    if ( !gets(buf) )
       exit(0);
while ( strcmp(buf, "_end_resp: ACTION") );

/* Look for end (blank line) of end of response.*/

do

    if ( !gets(buf) )
       exit(0);
while ( *buf );
```

```
/* Scan captured commands for _cmd:, _obj_id:, and */
/* _mod: lines and verify that " PO " is not present */
/* in the id. */

/* Pass empty SETs (from CANCEL in EDIT Dialog */
/* window). */

while ( 1 )
    {


    do
       {
       if ( !gets(buf) )
          exit(0);
       if ( !strncmp(buf, "_cmd: ", 6) )
          strcpy(cmd, &buf[6]);
       }

    while ( strncmp(buf, "_obj_id: ", 9) );

    modline = 0;

    do

      {
      if ( !gets(msgcmd) )
         exit(0);
      if ( !strncmp(msgcmd, "_mod: ", 6) )
         modline = 1;
      }
    while ( *msgcmd );

    if (!strstr(buf, " PO ") &&
      (modline || strcmp(cmd, "SET")))
       {

       if ( !strcmp(cmd, "SET") )
           {

          sprintf(msgcmd, "xterm -T "Error Message \
Dialogue\" -n \"Error\" -sf -fg black -bg white \
-fn 9x15bold -fb 9x15bold -geometry 100x5+200+500 \
-e dispmsg 'Altering %s is not allowed.'", buf+19);

           system(msgcmd);

           }
       printf("_error: \n\
_attr: errorSeverity E Error\n\
_attr: errorCode S \"C0001\"\n\
_attr: errorApplicationId S \"Customer\"\n\
_attr: errorFacility S \"Access Control\"\n\
_attr: errorInformation FS \"Creation/deletion/alter\
ation of this component is not allowed.\"\n\
\n");
       }
    printf("_end_resp:\n\n");
    fflush(stdout);
    }
}
```

In this example, component provisioning usually displays a separate window for an error message. In cases where this does not occur, the program brings up an xterm window. The xterm window then runs dispmsg to display the error message. Regardless of whether or not a window is used to display the error message, an error response is sent back to the Provisioning Command Filter API to block the command.

Event filters have an operator, P, to test for the presence of a particular attribute or category name. However, there is no operator to test for the absence of a particular attribute or category name. The program therefore creates a sieve to catch all data altering commands. It then checks the object IDs in the commands and selects only those that do not have the PO category name in the component ID. If the PO category is present in the component ID, an empty response (that is, just the end of response record) is sent back to the Provisioning Command Filter API to allow the command to continue.

Usage

For this customer application, to invoke Component Provisioning with the Provisioning Command Filter API from a menu, add the following to the pui command line in the toolset files:

```
-filter_appl limit2po -filter_width 100000
```

To compile the limit2po.c source, enter:

```
cc -O limit2po.c -o limit2po
```

Chapter 7
DPN PE Image Semantic Check Filter

This section describes a specific API command filter, the DPN PE Image Semantic Check Filter application. This section contains the following information:

- "API command filter DPN PE Image Semantic Check application" (page 77)
- "Usage of the DPN PE Image Semantic Check Filter application" (page 78)
- "Example" (page 78)

Note: This filter program is intended for use with DPN Provisioning only.

API command filter DPN PE Image Semantic Check application

This command filter application validates port service changes and PE_Loader Load file name changes to ensure that the configured port services are supported by the configured PE images. If a change would result in one or more port services that would not be supported by the PE image, the change is not made. In addition, an error message is displayed for each port service that would not be supported by the PE image.

The application notes any changes made to the Load file name in the PE_Loader component. If the Load file name has changed, the application checks to ensure that all the services provisioned for that PE under the PO component are supported by the new Load file name.

Another check is performed when services are provisioned under the PO component. When a new service is being provisioned, the application checks to ensure that the service is supported by the Load file name in the corresponding PE_Loader component.

Usage of the DPN PE Image Semantic Check Filter application

The API command filter DPN PE Image Semantic Check Filter application is called acf_dpn_image and is located in /opt/MagellanNMS/bin.

The usage of the DPN PE Image Semantic Check Filter application is:

```
pui -filter_appl “/opt/MagellanNMS/bin/acf_dpn_image <image filename>”
       [-filter_width 100000]
```

where each line in <image filename> has the format:

```
<image name> [<service name> <service name>...]
```

The usage rules are:

- <image filename> can end with the wildcard character (*)
- if <image filename> does not list an <image name>, all services are allowed
- if an <image name> does not list any corresponding service names, no services are supported by that <image name>
- for a given Load file name, the first entry in <image filename> that matches is used
- any blank lines and comment lines, which begin with an octothorpe (#), are ignored

Example

The contents of a sample image file are:

```
TRSNA.PIMG*  SNA Token_Ring
X25*         X25
RMOFFICE.*
```

In this example:

- for any version of the Load file name TRSNA.PIMG, no services other than SNA or Token_Ring are supported
- any Load file name beginning with X25 supports only the X25 service
- any Load file name beginning with RMOFFICE does not support any service
- for any images other than the above three, there are no restrictions on the services supported

Chapter 8
Post-Download: a specific filter

This section describes a specific API command filter, the post-download application. This section contains the following information:

Note: This filter is intended for use with DPN Provisioning only.

API command filter post-download application

Many activities can take place after a download has successfully completed, such as:

- populating the Network Reporting System (NRS)
- populating the Network Configuration Database (NCD)
- keeping a log file of who created the MCF

With a post-download activity in mind, you can make use of the post-download application in a program of yours.

This customer application provides an infrastructure that removes you from the details of the API command filter syntax. The post-download application provides 23 parameters, gathered from the DPN Component Provisioning authentication, upload, and download. The application keeps a record of what has been done, including who made the changes and the details of the upload and download.

This information can be passed as parameters to other programs that make use of this information. You can write programs that extract certain parameters to provide you with relevant information. For example, you might want to keep a log file of who created an MCF. By using the post-download application, you can easily obtain the userid of who made the changes, the MCF that was changed, where it was downloaded to, and the time at which the download occurred. Any messages that your program generates on standard output are displayed in a message window by the post-download application.

You can also use the post-download application to accomplish more complex actions. For example, after a download is complete, you can call a program to populate the Network Reporting System (NRS) and the Network Configuration Database (NCD). The NRS population application is a customer application that uses the post-download application. It is described in 241-6001-022 *Preside MDM Network Reporting System User Guide*.

Parameters of the post-download application

The post-download application gathers information from the authentication, upload, and download of a DPN Component Provisioning session. Once the download is complete, the application calls a program and passes the following information as positional parameters on the command line.

Table 1
Parameters of the post-download application

Position	Parameter	Notes
1	Userid	
2	NCS destination mnemonic	
3	NCS capability ID	
4	NCS password	
5	PM	
6	Upload location mode	Either USER_SPECIFIED or NMS_DISK.
(Sheet 1 of 2)		

Table 1 (continued)
Parameters of the post-download application

Position	Parameter	Notes
7	Upload location value	The PM from which the MCF was uploaded when the location mode is USER_SPECIFIED or empty for NMS_DISK.
8	Upload mode	One of ACTIVE, COMMITTED, KEYED, DATED, or USER_SPECIFIED.
9	Upload mode bundle or key or date	Empty for an upload mode of ACTIVE or COMMITTED.
10	Upload NAMSID	
11	Uploaded MCF	
12	Service data version of uploaded MCF	
13	Download type	Either COMPLETE or INCREMENTAL.
14	Download check	One of COMPLETE, INCREMENTAL, or MINIMAL_MCF.
15	Download location mode	Either USER_SPECIFIED or NMS_DISK.
16	Download location value	The PM to which the MCF was downloaded when the location mode is USER_SPECIFIED or empty for NMS_DISK.
17	Download mode	One of KEYED, DATED, or USER_SPECIFIED.
18	Download mode bundle or key or date	
19	Download NAMSID	
20	Download activation date	If this is set in download preferences.
21	Downloaded MCF	
22	Uploaded activation date	
23	Download activatable	One of YES or NO.
(Sheet 2 of 2)		

Usage of post-download application

The API command filter post-download application is called acf_post_download and is located in /opt/MagellanNMS/bin.

The usage of the post-download application is

```
acf_post_download [-backgrnd] <your program>
                  [<your options>]
                  [-h]
```

where:

- `-backgrnd` lets you run the post-download application in the background so that you can start another provisioning session
- `-h` lists the command line help

Making use of the post-download application

The post-download application provides you with parameters that you can pass to other programs. It is a specific API command filter.

The following steps will help you make use of the post-download application:

- To make use of the output of the post-download application, you write a program to extract the positional parameters you need. Add the processing that you want to do after a successful download, such as writing information to a log file, or populating NRS. Optionally, you can print progress and error messages to standard output. They will be displayed in a message window by the post-download application.
- To test the program, run Component Provisioning, for example:

```
pui  filter_appl "/opt/MagellanNMS/bin/
     acf_post_download <your program>
          [<your options>]"
     -filter_width 100000
```

- To activate the program permanently, you can add the filter to the PUIDpnCmd.cfg configuration file, as follows:

```
-filter_appl "/opt/MagellanNMS/bin/acf_post_download
        <your program> <your options>"
-filter_width 100000
```

For information on the configuration file, see “Configuring the Provisioning Command Filter API” (page 35).

Example

This example is a Bourne shell script.

You can create this script (let’s call it /tools/acf_log). It extracts three of the positional parameters from the post-download application. The variable user is the first parameter (userid), the variable pm is the fifth parameter (PM), and the variable dl_mcf is the twenty-first parameter, (shift 20), which is the downloaded MCF.

As soon as a successful download has completed, the script will append the echoed line to the log file, acf_pd_log. Note that the date is also added.

```
user=$1;
pm=$5;
shift 20;
dl_mcf=”$1”;

echo “PM: $pm, MCF: $dl_mcf, User: $user, Date: `date
     +%y-%m-%d`” >> /opt/MagellanNMS/data/log/
     acf_pd_log;
```

The line appended to the log file will look like the following:

```
PM: R34, MCF: MC.97040301.4034.0, User: myname, Date:
  97-03-12
```

To test the acf_log script, run Component Provisioning as follows:

```
pui  -filter_appl "/opt/MagellanNMS/bin
         /acf_post_download/tools/acf_log"
     -filter_width 100000
```

To activate the script permanently, add it to the configuration file, PUIDpnCmd.cfg located in /opt/MagellanNMS/cfg. Type the following in the configuration file:

```
-filter_appl "/opt/MagellanNMS/bin/acf_post_download
       /tools/acf_log"
-filter_width 100000
```

For a more complex example, see acf_pd_nrspop located in /opt/MagellanNMS/bin.

Appendix A
Provisioning Command Filter API object definitions

An online file containing information on the DPN Provisioning API object model is provided; this file is called /opt/MagellanNMS/lib/api/provapi.model. This file also contains object and attribute names as they appear in the Provisioning API model. For a description of this file, see 241-6001-204 *Preside MDM DPN Provisioning API Reference Guide*.

You can generate the Passport Provisioning API object model file using the ppmod command. For a description of this file, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*. For more information on the ppmod command, see 241-6001-207 *Preside MDM Passport Provisioning API Reference Guide*.

Appendix B Compliance statement

This appendix indicates how the Feature Specification of the Preside Multiservice Data Manager (MDM) Provisioning Command Filter API conforms to the API Primer.

This appendix contains the following:

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.
- N/A
  not applicable

API message types

The table “API message types” (page 90) lists the messages that are supported by the Provisioning Command Filter 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 Provisioning Command Filter API Provider.

Table 2
API message types

Message type	Direction	Supported
GET request	send/receive	Y
GET response	send/receive	Y
SET request	send/receive	Y
SET response	send/receive	Y
ACTION request	send/receive	Y
ACTION response	send/receive	Y
CREATE request	send/receive	Y
CREATE response	send/receive	Y
DELETE request	send/receive	Y
DELETE response	send/receive	Y
REGISTER request	send	Y
REGISTER response	receive	N
DEREGISTER request	receive	N
DEREGISTER response	send	N
EVENT-REPORT message	send	N
end-of-response message	send/receive	Y
error message	send/receive	Y
block message	send	N
version message	send	Y
end message	send/receive	Y

API message lines

The table “API message lines” (page 91) lists the message lines and keywords that are supported by the Provisioning Command Filter API. It describes overall support for the message line. The table “Optional message lines” (page 94) indicates the optional message lines that are supported by the Provisioning Command Filter 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 3
API message lines

Message line	Keywords	Supported
_action_type	upload, download, verify, discard, quit	Y
_attr		Y
_attr_id	general support ALL	Y Y
_block		N
_capability		N
_cmd	GET, SET, CREATE, DELETE, ACTION	Y (see also Table 3)
_echo	ON, OFF	Y
_end	general support USER_REQUEST FATAL_ERROR SYSTEM_SHUTDOWN LOST_CONNECTION LOST_SESSION	Y Y Y N Y N
_end_block		N
(Sheet 1 of 3)		

Table 3 (continued)
API message lines

Message line	Keywords	Supported
_error	general support	Y
	ACCESS_DENIED	N
	APPLICATION_ERROR	Y
	UNRECOVERABLE_ERROR	Y
	DUPLICATE_OBJECT	N
	INVALID_ACTION_TYPE	N
	INVALID_ATTRIBUTE_NAME	N
	INVALID_ATTRIBUTE_VALUE	N
	INVALID_FILTER	N
	INVALID_OBJECT_ID	N
	INVALID_OBJECT_TYPE	N
	INVALID_SCOPE	N
	LOGON_FAILS	N
	MISSING_ATTRIBUTE_VALUE	N
	MODIFICATION_ERROR	N
	NO_SUCH_OBJECT_ID	N
	OUT_OF_SEQUENCE	Y
_event_type		N
_filter	general support	N
	P	
	EQ	
	NE	
	LT	
	GT	
	LE	
	GE	
	LEFT	
	MIDDLE	
	RIGHT	
	IN	
	NOTIN	
_inv_id		Y

(Sheet 2 of 3)

Table 3 (continued)
API message lines

Message line	Keywords	Supported
_mod	general support ADD DEL DEF REP	Y Y Y Y Y
_obj_id		Y
_obj_class		Y
_password		N
_ref_obj_id		N
_scope	general support BASE NEXT ALL	C[1] Y Y Y
_sieve_id		Y
_sup_obj_id		N
_time		Y
_trace	ON, OFF	Y
_user_id		C[2]
_version		Y
(Sheet 3 of 3)		

1 See the table “Optional message lines” (page 94) for details.

2 Supported on REGISTER command event.

Table 4
Optional message lines

Message type	Message line	Supported
GET request	_inv_id _scope _filter _attr_id	Y Y N Y
GET response	_inv_id _attr	Y Y
SET request	_inv_id _scope _filter _mod	Y C[1] N C[2]
SET response	_inv_id _attr	Y N
ACTION request	_inv_id _scope _filter _attr	Y N N C[4]
ACTION response	_inv_id _attr	Y C[4]
CREATE request	_inv_id _obj_id _sup_obj_id _ref_obj_id _attr	Y Y N N Y
CREATE response	_inv_id	Y
DELETE request	_inv_id _scope _filter	Y C[4] N
DELETE response	_inv_id	Y
REGISTER request	_inv_id _user_id _password _attr	N C[4] C[4] C[4]
(Sheet 1 of 2)		

Table 4 (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	N/A
end-of-response message	_inv_id	Y
error message	_inv_id _attr	Y Y
(Sheet 2 of 2)		

1 The SET request only supports BASE scope.

2 A SET request must include at least one _mod: line.

3 The DELETE request only supports BASE scope.

4 Supported on command events.

API data types

The table “API data types” (page 96) lists the API data types supported by the Provisioning Command Filter 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 5
API data types

API data type	Supported
B (Boolean)	Y
I (Integer)	Y
H (Hexadecimal)	Y
D (Date/time)	N
S (String)	Y
FS (Formatted String)	Y
NI (Node Identifier)	Y
LI (Link Identifier)	N
E (Enumerated)	C[1]
SS (Sequence of Strings)	Y
SI (Sequence of Integers)	Y
SB (Sequence of Booleans)	Y
RS (Range of Strings)	N
RI (Range of Integers)	Y
RD (Range of Data)	N

1 The E (enumerated) data type is only in the errorSeverity attribute in the error messages.

Sieve object support

The table “Sieve object attributes” (page 97) lists the attributes of the sieve object that are supported by the Provisioning Command Filter API.

Table 6
Sieve object attributes

Attribute	Supported
general support	Y
sieveId	Y
eventFilter	Y
admState	Y
annotation	N
eventInfo	N
repOClass	N
repOid	N
repScope	N
repFilter	N
repInfo	N

Appendix C
Error messages

This appendix describes the error messages produced by the Provisioning Command Filter API. This appendix contains the following:

Response to undesired events

When an error occurs, one or more error messages are returned. If the error is not recoverable (by reissuing a corrected request from the customer application), an _end: line is also returned and the Provisioning Command Filter API process terminates.

If an error message received from a customer application has a severity of FATAL_ERROR, then the Provisioning Command Filter API Provider terminates after forwarding the message(s).

For grammar and syntax errors, only one error message is returned (for the first error), and the rest of the command message is purged.

The following error return codes are returned for errors detected by the Provisioning Command Filter API:

- INIT_ERROR
  occurs during Provisioning Command Filter API initialization. The version number is included on the _error: line. Since the version message is not sent (as initialization is not complete), the version number is included so the customer application can determine the format of the error message.

- SYNTAX_ERROR
  occurs when the request data does not conform to the syntax specified for the request.
- OUT_OF_SEQUENCE
  occurs when the request is not valid within the current context (for example, the Customer Application issues a GET Service Data request before the upload response is processed).
- APPLICATION_ERROR
  indicates that the request was syntactically correct, but semantically incorrect and the Field Access server has returned an error Protocol Data Unit (PDU) in response to the request.

If an error PDU is returned from the Field Access application, then the error response data consists of the text resulting from the formatting of the error qualifiers in the error PDU.

For lost sessions or connections, the API Provider returns an error message, followed by an end message, and then terminates.

Error messages

This section lists error messages generated by either the provisioning application or the Provisioning Command Filter API Provider.

> ***Note:*** In addition, error messages generated by User Interface, Provisioning API, Field Access, Envelope Access, and File Access Server error messages may be received and displayed by this application. The description of these error messages, however, is beyond the scope of this document.

Error messages from Provisioning Command Filter API Provider

The table “Error messages from the Provisioning Command Filter API Provider” (page 101) lists the error messages that can be generated by the Provisioning Command Filter API Provider. The $<n> items indicate values that are filled in at run time.

Table 7
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0001	`$0` A Customer Application has failed a request. The Customer Application supplies the text for this message. The meaning of this message and the action it requires is determined by the customer.
C0002	`Unknown connection status received: $0` Report this internal error to Nortel Networks.
C0003	`Unknown session status received: $0` Report this internal error to Nortel Networks.
C0004	`Unknown transaction status received: $0` Report this internal error to Nortel Networks.
C0007	`Fork failure: $0` It was not possible to start up the server. Either the UNIX process limit is too low and needs to be set to a higher value or the workstation is overloaded and the number of processes that are running needs to be reduced. Rescheduling may need to be considered. The UNIX system error message is substituted for $0 and should provide additional information.
C0008	`PDU failure.` Report this internal error to Nortel Networks.
(Sheet 1 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0009	`Lost connection.` This could be due to a software failure in the Field Access server. This should be reported to Nortel Networks. Alternatively, it may be due to a connectivity problem between the processes that need to be involved in the running of this application. Check with your local UNIX administrator to correct any local network or UNIX software problems and retry.
C0010	`Out of memory.` Either there is not enough real memory or swap space or both and the memory needs to be increased, or the workload on the workstation needs to be reduced or rescheduled.
C0011	`Invalid option: $0` Report this internal error to Nortel Networks.
C0012	`Unable to startup $0 with args $1 $2: $3` UNIX was not able to start up the server. Either the UNIX configuration is incorrect or the provisioning application was not installed properly. The Field Access image may be missing or it may have the wrong permissions. Make sure that UNIX has been configured properly. For example, ensure that the process limit is big enough. Also make sure that Architect has been installed properly. Then retry. The UNIX system error message is substituted for $3 and should provide additional information.
C0013	`Unable to initialize PROP PDO/PDU.` Report this internal software failure to Nortel Networks.
C0014	`Missing server name.` Report this internal software failure to Nortel Networks.
C0015	`Missing Field Access or Service Envelope Access log file name.` Report this internal software failure to Nortel Networks.
(Sheet 2 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0016	`Missing width.` Report this internal software failure to Nortel Networks.
C0017	`Invalid width: $0` Report this internal software failure to Nortel Networks.
C0018	`Unable to connect to server.` The server failed shortly after startup. Check the log file or the console or both for error messages to determine the cause of the failure.
C0019	`Unable to connect to node $0 port $1: $2` The Provisioning Command Filter API Provider was unable to place a stream socket call to the specified node and port. The message substituted for $2 should provide the required information to diagnose the problem. Make sure the server is active and accepting calls. Make sure that the node and port are specified correctly.
C0020	`Unexpected PDU reply code: $0.` Report this internal error to Nortel Networks.
C0021	`This provisioning command has been failed due to the premature termination of Customer Application "$0".` The user should try a download to save the changes made to this point and exit the provisioning session. The customer application should then receive the maintenance it requires. After that is done, a new provisioning session may begin. It may be necessary to remove the customer application from the list of applications interacting with the Provisioning Command Filter or to disable command filtering entirely.
C0022	`Missing node/IP address and port number.` The node/IP address and port number are missing from a -filter_serv option. Correct and retry.
(Sheet 3 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0023	`Missing port number.` The port number is missing from a -filter_serv option. Correct and retry.
C0024	`Missing UNIX command.` The UNIX command is missing from a -filter_appl option or macro invocation. Correct and retry.
C0025	`Unable to register service.` There is some problem with the Preside Multiservice Data Manager (MDM) Provisioning environment. Verify that MDM Provisioning has been installed properly and that at least a level 1 daemon is running.
C0026	`Missing service name parameter.` The service name to be used by the Provisioning Command Filter has not been specified. Correct and retry.
C0027	`Unable to open file $0: $1` The file specified for the "<" input redirection command could not be opened. The included UNIX error message should indicate the cause of the problem.
C0028	`Unable to popen command $0: $1` The Bourne shell command in the "@" command redirection escape could not be started. The included UNIX error message should indicate the cause of the problem.
C0029	`Unable to start up application $0: $1` The Bourne shell command in the -filter_appl option or ":" macro escape could not be started. The included UNIX error message should indicate the cause of the problem.
C0030	`Invalid port: $0` Only numeric digits are permitted in a port specification. Correct and retry.
(Sheet 4 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0031	`Unknown destination.` Destination IP address or mnemonic has not been typed correctly. Correct and retry.
C0032	`Missing command file name.` Command file name is missing from "<" input redirection command. Correct and retry.
C0101	`Improperly terminated string.` A string beginning with a quote must end with the same type of quote.
C0102	`Missing _action_type: line.` When the request is ACTION, there must be an _action_type: to specify the desired action. Correct and retry.
C0104	`Invalid compId type: $0.` Only "NI" is supported. Correct and retry.
C0105	`Invalid sieveId type: $0.` Only "I" is supported. Correct and retry.
C0106	`Missing $0.` Correct and retry.
C0107	`Invalid $0: $1.` Correct and retry.
C0108	`Improperly formed SET type value.` Correct and retry.
(Sheet 5 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0109	`Improperly formed RANGE type value.` Correct and retry.
C0110	`Unsupported $0: $1.` Not all valid API constructs are supported by the Provisioning Command Filter API. Retry the request excluding the unsupported construct.
C0111	`Extraneous data at end of line: $0.` Correct and retry.
C0112	`Invalid component id.` The component id does not have the correct form. Correct and retry.
C0113	`Missing object id.` Correct and retry.
C0114	`Re-specification of object id rejected.` Correct and retry.
C0115	`Service data GET request is not valid in the current context.` Service data GET requests are valid from the time a successful upload response is received until a download request is processed.
C0116	`Requested object(s) not found.` Correct and retry.
C0117	`Service data CREATE request is not valid.` Only sieves may be created.
C0118	`No Filter Elements were specified in the CREATE sieve request.` At least one filter element needs to be specified in a CREATE sieve request.
C0119	`Unknown sieve attribute: $0.` The only attributes permitted for sieves are eventFilter and admState.
(Sheet 6 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0120	`Only SS (sequence of string) is supported for the eventFilter attribute.` Correct and retry.
C0121	`Service data DELETE request is not valid.` Only sieves may be deleted.
C0122	`Service data SET request is not valid.` Only sieves may have their attributes changed.
C0123	`Invalid modification type: DEF` Sieve event filters do not have a default value.
C0124	`Modifications missing from SET request.` Correct and retry.
C0125	`Only BASE scope is supported for the SET request.` Correct and retry.
C0126	`Sieve $0 does not exist.` Correct and retry.
C0127	`Unsupported severity: $0.` Only Error and Fatal_Error severity in _error: messages from the customer application are supported at present. Correct and retry.
C0128	`The enable_sieves ACTION is not valid in the current context.` The enable_sieves ACTION request can be issued only once after the initial set of sieves has been created by the customer application. It cannot be issued again.
C0129	`Error message is not valid in the current context.` Error messages are not accepted during the initial sieve definition phase. Do not send error messages during this phase.
(Sheet 7 of 8)	

Table 7 (continued)
Error messages from the Provisioning Command Filter API Provider

Code	Description and possible solution
C0130	`End of Response message is not valid in the current context.` An end-of-response message is not valid during the initial state when the initial set of sieves is being created. Use the enable sieves ACTION request to end the initial sieve definition phase.
C0131	`Application "$0" has terminated before defining and enabling initial set of sieves.` The Customer Application has a programming error. Correct and retry.
C0132	`Passport category name hash table is full.` Report this internal error to your system administrator.
(Sheet 8 of 8)	

Phrases for substitution in error messages

The table “Phrases for substitution in error messages” (page 109) lists phrases that can be used for substitution in error messages.

Table 8
Phrases for substitution in error messages

Code	Description and possible solution
C0151	label of first line in request/response message
C0152	request code
C0153	end message line label
C0154	ACTION message line label
C0155	ACTION type
C0156	object ID keyword
C0157	object ID type
C0158	invoke ID
C0159	object class
C0160	scope
C0161	attribute name
C0162	attribute value type
C0163	attribute value
C0164	GET message line label
C0165	modification type
C0166	SET message line label
C0167	CREATE message line label
C0168	DELETE message line label
Co169	Enable Sieves message line label
C0170	Event Filter type
C0171	Event Filter attribute name
C0172	Event Filter operator
(Sheet 1 of 2)	

Table 8 (continued)
Phrases for substitution in error messages

Code	Description and possible solution
C0173	Event Filter attribute type
C0174	Event Filter attribute value
C0175	error severity attribute type
C0176	error code attribute type
C0177	error application id attribute type
C0178	error facility id attribute type
C0179	error information attribute type
C0180	error message attribute name
C0181	error severity attribute value
C0182	sieveId value
C0183	error message line label
C0184	error code attribute value
C0185	error application id attribute value
C0186	error facility id attribute value
C0187	error information attribute type
(Sheet 2 of 2)	

Index

Preside Multiservice Data Manager

Provisioning Command Filter API for DPN

Reference Guide

Release: R14.3

Publication: 241-6001-209
Document status: Standard
Document version: 15.1RSUP
Document date: August 2004
Printed in Canada