NORTEL NETWORKS

Preside Multiservice Data Manager

Passport Provisioning API

Reference Guide

241-6001-207

Preside Multiservice Data Manager

Passport Provisioning API

Reference Guide

Publication: 241-6001-207
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003

Printed in Canada

Publication history

December 2003

14.3RSUP Standard
Commercial availability

Contents

About this document

The following topics are discussed in this section:

Who should read this document and why

This document is for customers who use the Passport Provisioning Application Programming Interface (API) to write custom applications for the PresideMultiservice Data Manager (MDM) workstation.

What you need to know

To use the Passport Provisioning API, you need to know how to log on to the PresideMultiservice Data Manager (MDM) workstation.You must also be familiar with the UNIX operating system, UNIX C-Shell, and a UNIX editor such as vi.

Before using this document you should read 241-6001-200 *Preside MDM Application Programming Interface Primer*. This document provides an overview of the Application Programming Interface (API). The Passport Provisioning API conforms to the interface description contained in the *Programming Interface Primer.* The way in which the Passport Provisioning API conforms to the *Programming Interface Primer* is described in "Compliance statement" (page 91) in this document.

How this document is organized

241-6001-207 *Preside MDM Passport Provisioning API Reference Guide* contains the following sections:

- "Introduction" (page 15) describes the managed objects and API messages of the Passport Provisioning API.
- "Object model" (page 29) describes the containment hierarchy and the object classes of the Passport Provisioning API.
- "Using the Passport Provisioning API" (page 41) shows how to use the Passport Provisioning API by means of a set of examples.
- "API model difference tool" (page 83) shows how to compare two model files.
- "Compliance statement" (page 91) details how the Passport Provisioning API complies with the API Primer.
- "Error messages" (page 101) provides a list of error messages for the Passport Provisioning 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 PresideMultiservice 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:

- 241-6001-200 *Preside MDM Application Programming Interface Primer*
- 241-6001-303 *Preside MDM Administrator Guide*
- 241-6001-304 *Preside MDM Configuration Management Administrator Guide*

Chapter 1
Introduction

This section provides an introduction to the Passport Provisioning Applications Programming Interface (API). See the following sections for more information:

What the Passport Provisioning API is

The Passport Provisioning API is an ASCII interface that provides a customer-written software application with access to Passport provisioning and operational information on a Preside Multiservice Data Manager (MDM) workstation. The customer-written application can run on an MDM workstation or on another UNIX platform. With the Passport Provisioning API, the customer-written software application can be used to perform the following provisioning activities:

- upload provisioning information from a Passport
- create, delete, or modify provisioning information
- download modified provisioning information to a Passport

In addition to the above provisioning activities, the Passport Provisioning API lets you establish a non-provisioning session with the Passport.

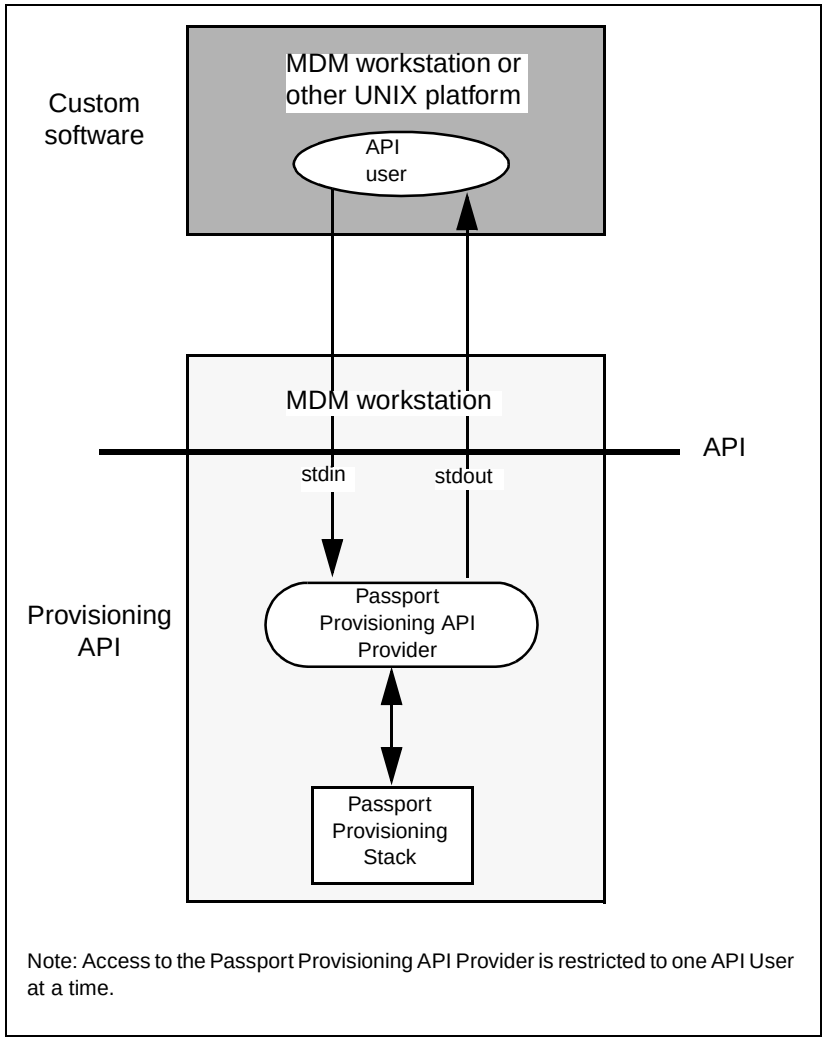
The Passport Provisioning API can also be invoked and used interactively from a data terminal. Using the Passport Provisioning API interactively is helpful for learning about the API before attempting to write a software application.

The figure “MDM workstation custom programming environment” (page 17) shows the communications with the Passport Provisioning API. In this figure, the API User is a customer-written application or a person at a data terminal who is using the Passport Provisioning API interactively. The Passport Provisioning API Provider is the MDM software that provides access to the Passport Provisioning Stack. The API is defined in terms of the messages passed between the API User and the Passport Provisioning API Provider.

The Passport Provisioning API reads service requests from standard input (stdin) or from an input file, converts them to internal format, and forwards them to the Passport Provisioning Stack. The stack executes the requests and returns responses to the Passport Provisioning API Provider. The Passport Provisioning API Provider writes responses to standard output (stdout). In addition, the API Provider sends a version message during initialization, and an end message during termination.

Piping between the API User and the Passport Provisioning API Provider must be established when the Passport Provisioning API is launched. This piping is external to the Passport Provisioning API. Possible methods of establishing piping are explained in the 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 1
MDM workstation custom programming environment



Passport Provisioning API Model Generator

The Passport Provisioning API Model Generator provides a means to generate a set of files containing Passport object and attribute descriptions from a Passport activation file.

For each version of a Passport software load, there is corresponding version of Passport activation file. A Passport activation file can be considered as a schema that contains the descriptions and characteristics of the objects and attributes in a Passport switch for a particular Passport software load.

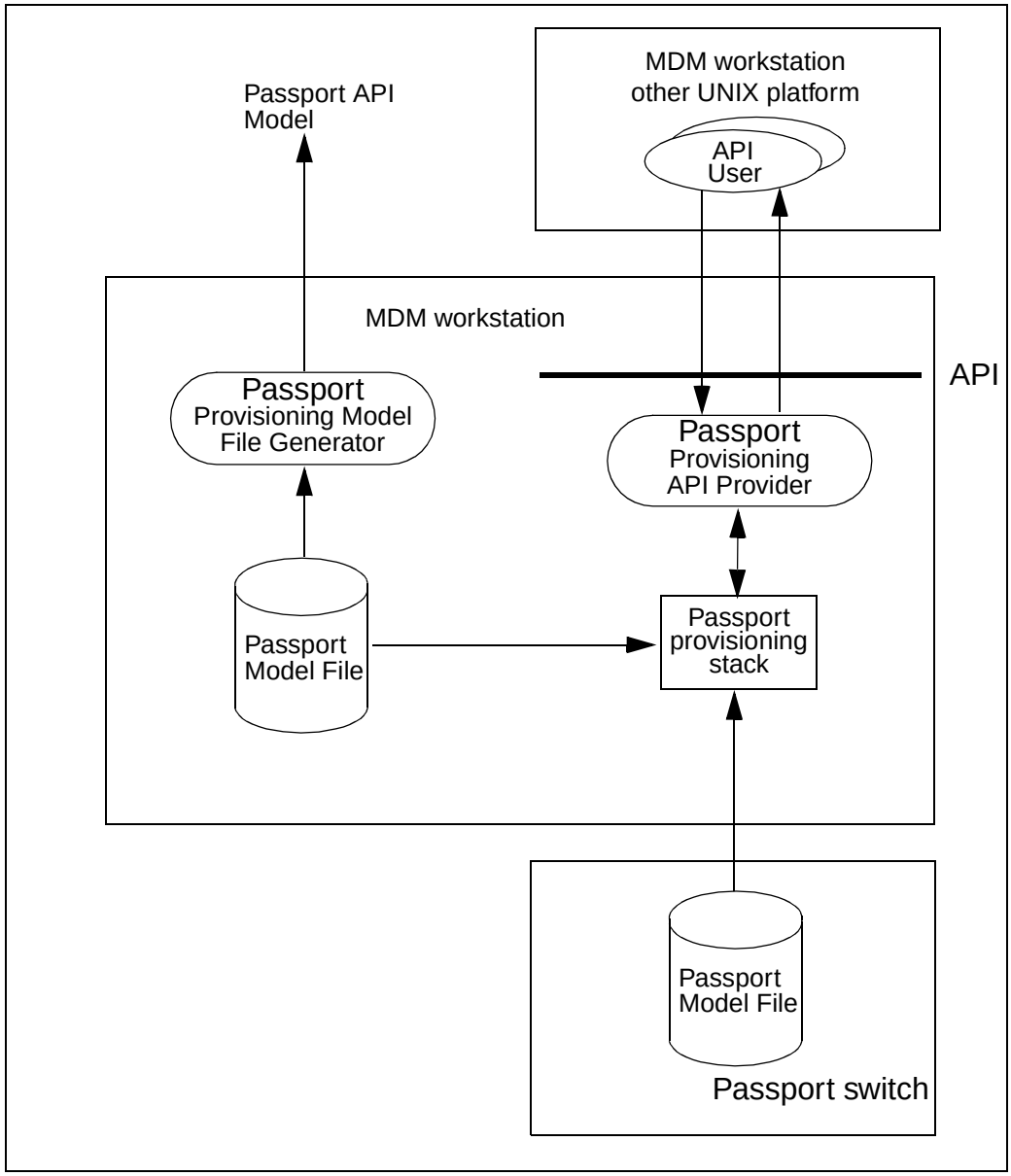
The Passport Provisioning API Model Generator can be used to generate a Provisioning API Model from the active version of the Passport activation file that is stored on the NMS disk. See the figure "Generating Provisioning API Model files from Passport Model files" (page 19).

> ***Note:*** The Passport Provisioning API Model consists of a set of three files.

The main reason for generating a set of Passport Provisioning API Model files is that unlike the Passport activation file, the model files provide efficient access to the following information:

- the distinguished name, attribute syntax, attribute modification and legal values for an object
- the hierarchical structure of provisionable objects for a Passport

Figure 2
Generating Provisioning API Model files from Passport Model files



Data types

The Passport Provisioning API supports the following data types:

- NI (Node ID)
- I (Integer)
- L (Long Integer)
- S (String)
- SS (Sequence of Strings)

For an explanation of these datatypes, see the 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Sieves and event report types

The Passport Provisioning API does not support sieves or event reports.

Message types

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

The Passport Provisioning API supports the following types of messages:

- Version
- Request
- Response
- End of response
- Error
- End

The following paragraphs summarize each of these type of messages.

Version message

The Passport Provisioning API Provider returns a version message to the API User on successful startup of the Passport Provisioning API. The version message indicates the release of the Passport API software that has been successfully started up.

Version message format

The general format of a version message is as follows:

```
_version: <version number> <version string>
```

where:

`<version number>`

is the software release number of the Passport Provisioning API

`<version string>`

is the text string: _version <software release> Passport Prov API Copyright Nortel Networks <year>

Request message

The API User sends a request message to the Passport Provisioning API Provider. The types of request messages supported by the Passport Provisioning API are as follows:

REGISTER	establishes the necessary authorization for performing provisioning activities
DEREGISTER	discards the authorization established with the REGISTER command
GET	retrieves provisioning data
SET	changes the values of the attributes for a provisioned component
CREATE	creates a provisioned component

DELETE	deletes a provisioned component
ACTION	orders that a command be performed on a specified object. The commands are: upload, quit, download, discard, and verify.

Request message format

The general format of a request message is as follows:

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

where:

`<command keyword>`

is one of: REGISTER, DEREGISTER, GET, SET, CREATE, DELETE, or ACTION

`<invoke ID>`

is an optional integer supplied by the API User. This integer is returned in all responses to a request.

`<request data line>`

is information that further defines the request

ACTION request message

The ACTION request message includes an action keyword and has the following general message format:

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

where:

`<action keyword>`

is one of: Upload, Download, Quit, Discard, or Verify. The purposes of these action keywords are described in “ACTION requests” (page 71).

Response message

The Passport Provisioning API Provider returns one or more response messages to the API User as a result of handling a request message. The last response message in the series of response messages is an end of response message. The general format of a response message is as follows:

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

where:

`<invoke ID>`

is an optional integer that was originally supplied by the API User. This integer identifies the request to which the response applies.

`<response data line>`

is information that further defines the response. If the request was successful, this line contains the data requested. If the request was unsuccessful, this line contains text that indicates the nature of the failure.

End of response message

The Passport Provisioning API Provider returns an end of response message to the API User as the last response message in a series of response messages to a given request message. The general format of an end of response message is as follows:

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

where:

`<invoke ID>`

is an optional integer that was originally supplied by the API User. This integer identifies the request to which the end if response message applies.

`<request>`

is USER_REQUEST to indicate that the API User requested that the request be aborted, or is the type of request message that initiated the series of response messages: REGISTER, DEREGISTER, GET, SET, CREATE, DELETE, or ACTION

`_time: <year> <month> <day> <hour> <minute> <second>`

is a time stamp that indicates when the Passport Provisioning API Provider sent the end of response message

Error messages

The Passport Provisioning API Provider returns one or more error messages to the API User to indicate a failure in the processing of a request. The last error message returned is an end of response message. The general format of an error message is as follows:

```
[_inv_id: <invoke ID>]
_error: <API error code>
_attr: errorSeverity E <severity>
_attr: errorCode S <application error code>
_attr: errorApplicationId S <application 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 an optional integer that was originally supplied by the API User. This integer identifies the request to which the error message applies.

`<API error code>`

is a keyword that indicates the general category o f the error. This keyword is one of: INIT_ERROR, SYNTAX_ERROR, NOT_SUPPORTED, OUT_OF_SEQUENCE, or APPLICATION ERROR

For the SYNTAX_ERROR message, the Passport Provisioning API stops processing and purges the request.

`<severity>`

is the severity of the error. The severity is one of: Information, Warning. Error, or Fatal_Error. The Passport Provisioning API handles error messages as follows:

For error messages with a severity of Information or Warning, the Passport Provisioning API completes processing, sends a response message, then sends an end of response message.

For error messages with a severity of Error, the Passport Provisioning API does not process the request, sends the error message(s), then sends an end of response message.

For error messages with a severity of Fatal_Error, the Passport Provisioning API terminates with an end message.

`<application error code>`

is text associated with the error code that is returned by the Passport Provisioning Stack. For a list of application error codes and their meanings, see “Error messages” (page 101).

`<application ID>`

is an application identifier that specifies that the Provisioning Stack generated the error message.

`<facility>`

is a facility code that specifies the software facility within the application layer of the Provisioning Stack that generated the error.

`<textual error message>`

is additional text that describes the error.

`<extra data value>`

is extra data associated with the error.

End message

The Passport Provisioning API Provider or the API User sends an end message to terminate an API session.

An end message sent by the API User has the following format:

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

An end message sent by the Passport Provisioning API Provider has one of the following formats:

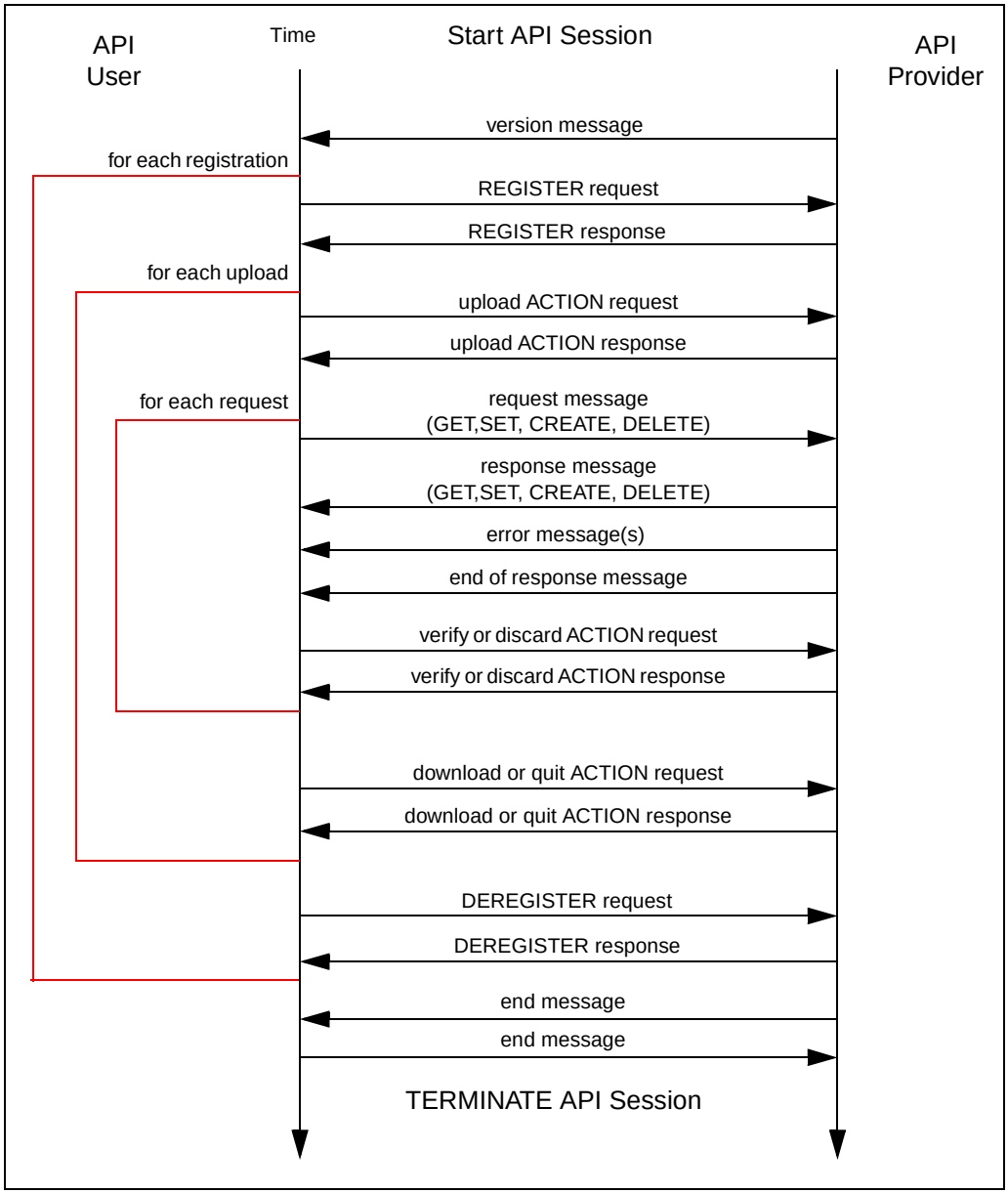
- ```
  _end: USER_REQUEST
  <blank_line>
  ```
- ```
  _end: FATAL_ERROR
  <blank line>
  ```

The first of these two messages indicates that the request was aborted at the request of the API User. The second indicates that Passport Provisioning API Provider aborted the request because one of the following unrecoverable errors has occurred:

Startup failure	The Passport Provisioning API was unable to start up an FFA/FPA process. Since the _version: message has not been output yet the version is included on the "_error: INIT_ERROR" line.
PDU failure	The Passport Provisioning API was unable to send or receive a PDU to or from the FFA process. A PDU is a specially formatted IPC message exchanged by provisioning applications. This failure might be an indication of a software problem.
Lost session	The FFA process aborted the session. This failure might be an indication of a software problem.
Lost connection	The Passport Provisioning API is unable to communicate with the FFA process. This failure might be an indication of a software problem.
Out of memory	An attempt to allocate memory failed. This failure indicates that a runaway process is using up memory or that insufficient memory is provisioned on the workstation.

The figure "Time line diagram" (page 28) shows an overview of the session events. It represents the normal sequence of messages between the API Provider and the API User. For more details on the message types, see the 241-6001-200 *Preside MDM Application Programming Interface Primer*.

Figure 3
Time line diagram



Chapter 2
Object model

This section describes the containment hierarchy and the object classes for the Passport Provisioning API.

The Passport Provisioning API object model cannot be included in this section because it is too large and is dynamic in nature. Information is provided about the object model contained in a set of files that you can generate from a Passport activation load by means of the Passport Provisioning Model Generator.

See the following sections for more information:

- "Object model files" (page 29)
- "Containment hierarchy" (page 30)
- "Object classes" (page 33)
- "Attributes" (page 35)
- "Hierarchy of objects" (page 39)

Object model files

A Passport Provisioning Model Generator is supplied with the Passport Provisioning API software. This generator creates three files from a Passport activation file. Together, these three files contain a complete Passport

Provisioning Object Model. You can use the information in this model to help you formulate request messages for performing provisioning operations on Passport switches. The files are as follows:

<version>.comps, the objects file

This file defines of the classes of objects and the message types supported for each object.

<version>.attributes, the attributes file

This file contains the attribute definitions for the objects.

<version>.hier, the hierarchy file

This file contains a hierarchy of the objects and their attributes.

where:

<version>

is the version of the file. Although you can specify a different version, the default version is the same as the version of the Passport activation file from which the file is created.

For the instructions to generate the three files that contain the Passport Provisioning Object Model, and when to generate them, see "Generating a Passport Provisioning API Model" (page 43).

Containment hierarchy

The figure "Example of a service data containment hierarchy" (page 33) shows an example of how service data components for Passport switches are organized into a containment hierarchy, starting at the top EM (Enterprise Module) component in the hierarchy and continuing down to its service data components.

The following paragraphs contain an explanation of the components in the service data hierarchy shown in the figure "Example of an array" (page 38).

Service data component names

Each service data component in the hierarchy has a name and a key value. For example, in the figure “Example of a service data containment hierarchy” (page 33), FrameRelayUni (FrUni) is the component name and 130 is the key value that identifies the component called: FrUni.

The following describes three ways to identify a service data component:

- with the abbreviated name of the component and a key value. For example, FrUni 130.
- with the full component name, also called the prompt, and a key value. For example, FrameRelayUni 130.
- with a component attribute identifier (ID) and a key value. For example, 279 130.

Distinguished service data component names

The distinguished name of a service data component consists of the component name and its key value, along with the component names and key values of all other components above it in the containment hierarchy. For example, in the figure “Example of a service data containment hierarchy” (page 33), the distinguished name of component Dna is:

```
EM Passport1 FrUni 130 Dna NULL
```

where:

`EM` and `FrUni`

are the component names of all components that are above component Dna in the hierarchy

`Passport1, 130,` and `NULL`

are the key values

You can identify components in a distinguished name using any one of the following methods:

- with component attribute identifiers (IDs) only
- with abbreviated names of the components only
- with the full names (prompts) of the components only

- with a mix of abbreviated names and full names of the components

  Note: You can only mix abbreviated names and full names in a distinguished name. You cannot mix IDs with abbreviated names or full names.

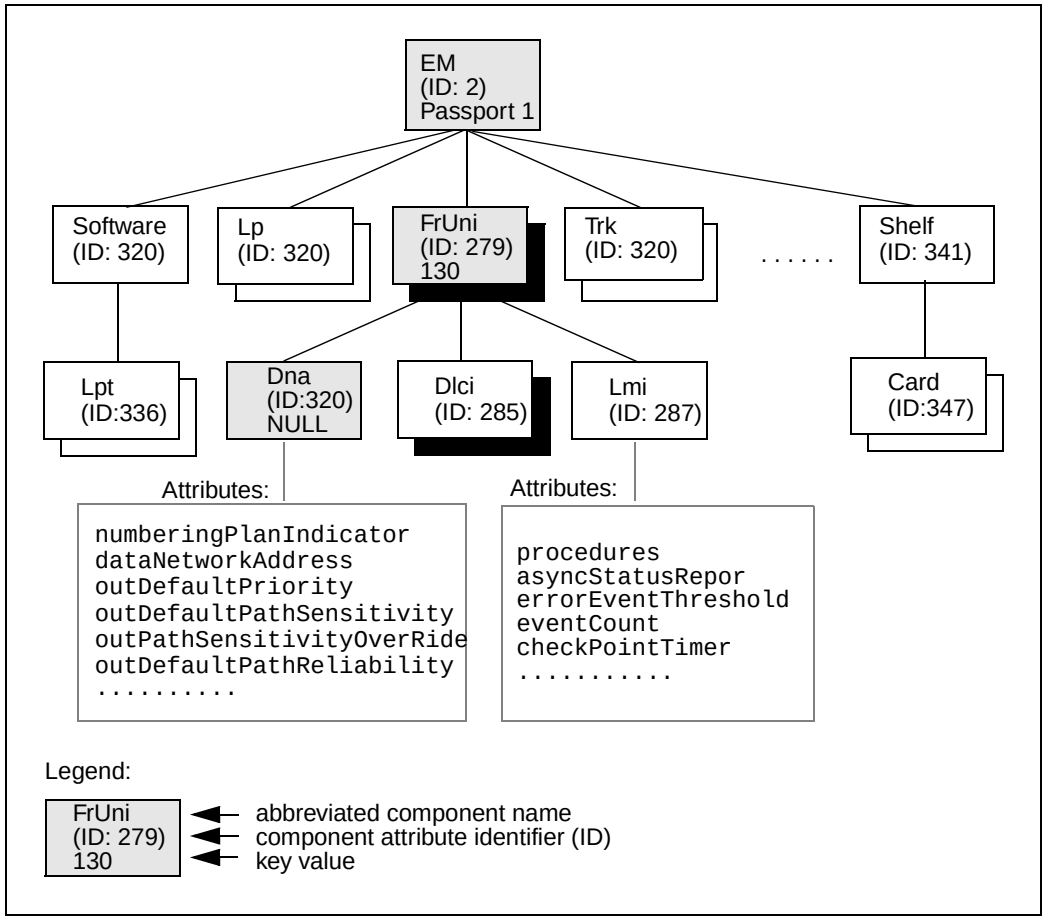
The following are examples of the three methods for writing the distinguished name of the same service data component:

- 2 Passport1 279 130 320 NULL
  Method: component attribute identifiers (IDs) only
- EM Passport1 FrUni 130 Dna NULL
  Method: abbreviated names only
- EM Passport1 FrameRelayUni 130 DataNetworkAddress NULL
  Method: full names (prompts) only
- EM Passport1 FrUni 130 DataNetworkAddress NULL
  Method: a mix of abbreviated component names and full names (prompts)

Attributes

These are the attributes (fields) of the service data components. For example, some of the attributes of component Dna are: numberingPlanIndicator, outDefaultPriority, and outDefaultPathReliability.

Figure 4
Example of a service data containment hierarchy



Object classes

The Passport Provisioning object classes and the actions that can be performed on them are specified in a file called <version>.comps, where <version> is the version of the file. You must create this file from a Passport activation file with the Passport Provisioning API Model Generator. For the instructions to generate this file, see “Generating a Passport Provisioning API Model” (page 43).

The classes and actions are specified in the following format:

```
OBJECT CLASS: <prompt>
ID: <id>
[ABBREVIATION: <abbreviation>]
*key value: <value>
IS MEMBER OF: <distinguished paths>
MUST CONTAIN: {<attributes and components that object
of this class must have>}
MAY CONTAIN:{<attributes and components that object of
this class may have>}
```

The following example shows how object class IpInterfaceOverVc is defined in a sample objects file.

```
OBJECT CLASS: IpInterfaceOverVc
ID          : 1465
ABBREVIATION: Ipivc
*key value: NULL
IS MEMBER OF: EM
MAY CONTAIN: {
 1468 (maxLcn) maximumNumberOfLcn             GET SET,
 161 (Dna) DataNetworkAddress        CREATE DELETE GET
 1467 (ipa) ipAddress                      GET SET,
 1470 (Dr) DefaultRoute              CREATE DELETE GET
}
```

The actions supported are also specified in the object class information file in the following format:

```
ACTION TYPE: <action type>
WITH ATTRIBUTE: {<attribute that actions of this type
may have>}
RESPONSE ATTRIBUTES: {<attributes that response to
this type of action may have>}
```

The following example shows how the upload action is defined in a sample objects file.

```
ACTION TYPE: upload
WITH ATTRIBUTES:     {
                     accessmode
```

```
                        view
                         }
RESPONSE ATTRIBUTES: {
                          viewname
                          sdversion
                          swversion
                        }
```

Attributes

The attributes associated with Passport Provisioning objects are specified in a file called <version>.attributes, where <version> is the version of the file. You must create this file from a Passport activation file with the Passport Provisioning Model Generator. For the instructions to generate this file, see “Generating a Passport Provisioning API Model” (page 43).

The attributes for an object class are specified in the following general format:

```
ATTRIBUTE: “<prompt>”
ID: <id>
[ABBREVIATION: <abbreviation>]
IS MEMBER OF: [distinguished paths]
SYNTAX: <attribute value type>
LEGAL VALUES: <legal value for the defined attribute>
[DEFAULT: <default value>]
[*Format: <description>]
```

The following example shows how the accountClass attribute is specified in a sample <version>.attributes file.

```
ATTRIBUTE   : accountClass
ATTRIBUTE ID: 200
ABBREVIATION: acl
IS MEMBER OF: EM/FrNni/Dna
              EM/FrUni/Dna
SYNTAX      : I
LEGAL VALUES: 0..255
DEFAULT     : 0
```

Four main types of structured data attributes are used for Passport provisioning. These are: lists, vectors, replicated data, and arrays. The format of each of these attributes is described in the following paragraphs.

List attributes

A list is a data structure that contains a variable number of values. Individual items cannot be addressed in a list. You can retrieve an entire list, replace an entire list, add a value to a list, delete a value from the list or set an entire list to a null (an empty list).

A list type attribute is specified in the following format:

```
ATTRIBUTE: “<prompt>”
ID: <id>
[ABBREVIATION: <abbreviation>]
IS MEMBER OF: [distinguished paths]
SYNTAX: <attribute value type>
LEGAL VALUES: <legal value for the defined attribute>
[DEFAULT: <default value>]
[*Format: <format description>]
*Legal modification: REP, DEF, ADD, DEL
```

The following example shows how a list attribute is defined in a sample attributes file.

ATTRIBUTE : avList
ATTRIBUTE ID: 323
ABBREVIATION: avl
IS MEMBER OF: EM/Sw
SYNTAX : S
*Format : 1...25 ASCII characters
*Legal modification : REP, DEF, ADD, DEL

Vector and replicated data attributes

A vector is a data structure that contains a fixed number of values. You can access individual items in a vector by means of index that consists of a single term. An individual value can be retrieved or replaced in a vector. An individual value in a vector that can be addressed by means of an index is called a cell.

Replicated data is a data structure that is identical to a vector, except that unlike a vector which contains a fixed number of values, you can add items to replicated data.

Vector and replicated data attributes are specified in the following format:

```
ATTRIBUTE: “<prompt>”
ATTRIBUTE ID: <id>
[ABBREVIATION: <abbreviation>]
IS MEMBER OF: [distinguished paths]
SYNTAX: <attribute value type for the cell>
LEGAL VALUES: <legal value for the cell>
[DEFAULT: <default value>]
[*Format: <format description>]
*Legal modification: DEF /* for REPLICATED data */
*With cell format: <prompt>[<indexType> <INDEX1>]
*Index 1 legal value: <legal value for the first index>
*Cell legal modification: REP /* for VECTOR */
*Cell legal modification: REP, ADD, DEL /* for
REPLICATED data */
```

The following example shows how a vector data attribute is defined in a sample attributes file.

```
ATTRIBUTE   : seizeCode
ATTRIBUTE ID: 2130
IS MEMBER OF: EM/Vs/Frame
SYNTAX      : I
LEGAL VALUES: 0..1
*With cell format    : seizeCode[S <INDEX1>]
*Index 1 legal value : {“A”,”B”,”C”,”D”}
*Cell legal modification  : REP
```

The following example shows how a replicated data attribute is defined in a sample attributes file.

```
ATTRIBUTE   : natlFax
ATTRIBUTE ID: 235
ABBREVIATION: nfa
IS MEMBER OF: EM/IpiFr/Lcn/Dc
SYNTAX      : S
*Format:  0...256 Hexadecimal digits
*Legal modification: DEF
*With cell format       : nfa[I <INDEX1>]
*Index 1 legal value    : 1..1
*Cell legal modification: REP, ADD, DEL
```

Array attributes

An array is a data structure that consists of a table of values. A value anywhere in the array can be accessed by an index consisting of two terms. An example of an array used to store packet window sizes for an X.25 link is shown in the following figure. This array consists of 12 rows and 16 columns of packet window sizes, and each item in the array can be accessed by a two term index consisting of the packetSize (one of: 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 32768, or 65535) and a throughput Class (an integer from 0 to 15).

A value anywhere in the array that can be address by means of a two-term index is called a cell.

Figure 5
Example of an array

	throughputClass															
packetSize	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
32	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
64	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
128	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
256	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
512	6	6	4	6	6	6	6	6	6	6	6	6	6	6	6	6
1024	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
2048	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4096	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
8192	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
32768	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
65535	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

An array attribute is specified in the following format:

```
ATTRIBUTE: “<prompt>”
ATTRIBUTE ID: <id>
[ABBREVIATION: <abbreviation>]
IS MEMBER OF: [distinguished paths]
SYNTAX: <attribute value type for the cell>
LEGAL VALUES: <legal value for the cell>
[DEFAULT: <default value>]
[*Format: <format description>]
*With cell format:
<prompt>[<indexType> <INDEX1>][<indexType> <INDEX2>]
```

```
*Index 1 legal value: <legal value for the first index>
*Index 2 legal value: <legal value for the second
index>
*Cell legal modification: REP
```

The following example shows how an array attribute is defined in a sample attributes file.

```
ATTRIBUTE   : windowSize
ATTRIBUTE ID: 245
ABBREVIATION: wins
IS MEMBER OF: EM/Mod/Vcs
SYNTAX      : I
LEGAL VALUES: 1..63
*With cell format    : windowSize[S <INDEX1>][I
<INDEX2>]
*Index 1 legal value :
{“16”,”32”,”64”,”128”,”256”,”512”,”1024”,”2048”,
                        “4096”,”8192”,”32768”,”65535”}
*Index 2 legal value : 0..15
*Cell legal modification  : REP
```

Hierarchy of objects

The hierarchy structures associated with Passport Provisioning objects are specified in a file called <version>.hier, where <version> is the version of the file. You must create this file from a Passport activation file with the Passport Provisioning Model Generator. For the instructions to generate this file, see “Generating a Passport Provisioning API Model” (page 43).

The hierarchy structure associated with an object is specified in the following general format:

```
<indentation> <id> [(<abbreviation>)] <name>
```

The following example shows how the hierarchy is defined in a sample hierarchy file. Because the hierarchy is large, this example only shows the first few lines of the file.

```
*2 EM
*--1352 (Npi) NumberingPlanIndicator
*----1360 totalDnas
*----1353 Dna
*------1355 destinationName
```

```
*--117 (Col) Collector
*----119 (agentQ) agentQueueSize
*----120 (times) collectionTimes
*----123 (Sp) Spooler
*------125 (spool) spooling
*------126 (maxFile) maximumNumberOfFiles
*--2131 (NS) NetworkSynchronization
*----2139 (priRef) primaryReference
*----2140 (secRef) secondaryReference
*----2141 (tertRef) tertiaryReference
..........
```

Chapter 3
Using the Passport Provisioning API

This section shows you how to use the Passport Provisioning API by means of a set of examples. See the following sections for more information:

Task list for using the Passport Provisioning API for the first time

The following list shows the tasks you need to perform to run the Passport Provisioning API for the first time. This list also contains references to sections that describe how to perform each of these tasks.

- Ensure that the prerequisites for using the Passport Provisioning API are performed.

  See “Prerequisites for running the Provisioning API and the API Model Generator” (page 43).

- Install and configure the Passport Provisioning API and the Passport Provisioning Model Generator.

  See “Installing and configuring the Passport Provisioning API” (page 43).

- Generate the Passport Provisioning Model using the Passport Provisioning Model Generator.

  See “Generating a Passport Provisioning API Model” (page 43).

- Set up piping, if required.

  See “Setting up piping” (page 48).

- Start the Passport Provisioning API.

  See “Starting the Passport Provisioning API” (page 48).

- Run the session interactively or from a command file.

  See “Running a session” (page 51).

- Terminate Passport Provisioning API Access.

  See “Terminating Passport Provisioning API Access” (page 54).

Prerequisites for running the Provisioning API and the API Model Generator

Before you can use the Passport Provisioning API or the Passport Provisioning API Model Generator, you need to do the following:

- install the Passport Provisioning API and Passport Provisioning API Model Generator software on the Preside Multiservice Data Manager (MDM) workstation.
- set up the Passport groups and members in file /opt/MagellanNMS/cfg/HGDS.cfg
- start the following servers on the MDM workstation and ensure they remain running:
  - Host Group Directory Server (HGDS)
  - Passport Data Translator Manager (FDTM)
  - Connection Manager (CM)

Installing and configuring the Passport Provisioning API

The Passport Provisioning API and the Passport Provisioning API Model Generator are installed as part of the Architect software package. For the instructions to install the software, see 241-6001-100 *Preside MDM Installer Guide*.

Generating a Passport Provisioning API Model

The Passport Provisioning API Model Generator provides a means to generate a set of three files from a Passport activation file that contain Passport object, attribute, and hierarchy descriptions. Together these three files constitute the Passport Provisioning Object Model.

You need to create a set of these model files before you use the Passport Provisioning API to perform provisioning activities on a Passport switch that is running a new release of Passport software.

There are two ways to use the Passport Provisioning API Model Generator: in prompt (command line) mode and in interactive mode. Because the interactive mode provides a Query option, you can also use the interactive mode to query an attribute or object class; this is something you cannot do in the prompt (command line) mode.

The syntax of the command to run the Passport Provisioning API Model Generator is as follows:

- to obtain information about command line syntax:

  /opt/MagellanNMS/bin/ppmod -h

- to run the generator interactively:

  /opt/MagellanNMS/bin/ppmod

- to run the generator without prompting:

  /opt/MagellanNMS/bin/ppmod -v <version> [-m][-a][-p] [-c]

where:

`-h` displays the syntax of the ppmod command.

`-v` specifies that a version is to be entered. This parameter is mandatory.

`<version>` specifies the version of the Passport activation file from which the model files are to be generated, for example, AC0009e or AC0012c. This parameter is mandatory.

The ppmod command uses the specified version to construct the name of the Passport activation file that it will load into shared memory and will make into the active file. It also generates the model files from this file.

The Passport activation files are located in directory /opt/MagellanNMS/cfg/PassportSchema. These files are named PfmsCF<version>.act.

`[-m]` generates the object classes information. The output filename is <version>.comps.

`[-a]` generates the attributes information. The output filename is <version>.attrs.

`[-p]` generates provisioning hierarchy information. The output filename is <version>.hier.

`[-c]` include operational components/attributes information. The output filenames are <version>.compa_all, <version>.attrs_all, and <version>.hier_all.

If you specify <version> without any of options [-m], [-a], or [-p], all three files are generated.

Generating the files interactively

The interactive mode allows you generate object class, attribute, or hierarchy files. Because the interactive mode provides a Query option, you can use it to find out information about an attribute or an object class. Because the Query option also supports wild card characters, you can use the interactive mode to generate information for all objects or attributes that have a name, an ID, or an abbreviation that matches a specified pattern.

The following is an example of an interactive session. In this example, AC0009e is used as the version.

1 Enter the ppmod command without options to run the generator interactively.

```
> /opt/MagellanNMS/bin/ppmod
```

All of the available versions in directory /opt/MagellanNMS/cfg/PassportSchema are displayed as follows:

```
Available versions under '/opt/MagellanNMS/data/'
directory:AC0011c          AC0009e          AC0012c
```

The following prompt is displayed on the screen:

```
Please enter the version you want [Hit <Return> to
Quit]: AC0009e
```

2 If you press the return key without entering a version, the generator terminates. Otherwise, the following prompt is displayed:

```
Do you want to include operational components and
attributes? y/[n]:
```

If you enter n or press the return key, only the information for provisionable components and attributes is included in the output files.

If you enter y , the information for operational components and attributes is also included in the output files. The output filenames are suffixed with _all.

The following message appears, followed by a pause while the file is being loaded:

```
Activating model file opt/MagellanNMS/cfg/
PassportSchema/pfmsCFAC0009e.act
```

The following menu is displayed:

```
Options:
0: Quit
1: Output object classes
2: Output attributes
3: Output provisioning hierarchy
4: Query

Please select the option you want:
```

3 Enter 1 to generate a file containing the object classes. The following prompt is displayed:

```
Please enter the output filename [AC0009e.comps]:
```

4 If you press the return key without entering a path and filename, the generator produces an object classes file in the current directory, and it has the default name <version>.comps. In this example, use the default filename and the current directory.

The generator creates the output file and displays the following response, menu, and prompt:

```
File AC0009e.comps (object class information) is
generated.

Options:
0: Quit
1: Output object classes
2: Output attributes
3: Output provisioning hierarchy
4: Query

Please select the option you want:
```

5 Enter 2 to generate a file containing the object attributes. The following prompt is displayed:

```
Please enter the output filename [AC0009e.attrs]:
```

6 Press the return key without entering a filename. The generator produces an attributes file in the current directory, and it has the default name <version>.attrs.

The generator creates the output file and displays the following response, menu, and prompt:

```
File AC0009e.comps (object class information) is
generated.

Options:
0: Quit
1: Output object classes
2: Output attributes
3: Output provisioning hierarchy
4: Query

Please select the option you want:
```

7 Enter 3 to generate a file containing the hierarchy information. The following prompt is displayed:

```
Please enter the output filename [AC0009e.hier]:
```

8 Press the return key without entering a filename. The generator produces a hierarchy file in the current directory, and it has the default name <version>.hier.

The generator creates the output file and displays the following response, menu, and prompt:

```
File AC0009e.hier (provisioning hierarchy) is
generated.

Options:
0: Quit
1: Output object classes
2: Output attributes
3: Output provisioning hierarchy
4: Query

Please select the option you want:
```

9 Enter 4 to query information. The following prompt is displayed:

```
Please enter the name, id or abbreviation:
```

10 Enter *Code* to search for all entries that have the string Code in them, regardless of the characters ahead or after the string .

Note: The asterisks (*) are the wildcard characters.

The following prompt is displayed:

```
Please enter the output filename [stdout]:
```

11 If you press the return key without entering a path and filename, the generator displays the results of the query on the screen. In this example, press the return key and display the results on the screen.

The results of the query are displayed on the screen as follows:

```
ATTRIBUTE   : idleCode
.......
.......

ATTRIBUTE   : networkIdCode
.......
.......

Number of records that match '*Code*': 2
```

If the output file cannot be opened, the generator displays an error message followed by the option menu.

Setting up piping

You need to establish the necessary piping between the Passport Provisioning API Provider and the API User at the time the API provider is started. Establishment of the necessary piping is external to the API. For a description of how to set up piping, see the 241-6001-200 *Preside MDM Application Programming Interface Primer.*

It is not necessary to set up piping if you are going to use the Passport Provisioning API interactively on a Preside Multiservice Data Manager (MDM) workstation.

Starting the Passport Provisioning API

The syntax of the pprovapi command to start the Passport Provisioning API is as follows:

- to obtain information about command line syntax:
  `/opt/MagellanNMS/bin/pprovapi -h`

- to run the Passport Provisioning API:
  /opt/MagellanNMS/bin/pprovapi
  [-id]
  [-nouserid]
  [-width <width>]
  [-echo]
  [-trace]

where:

`-h` displays the syntax of the pprovapi command.

`[-id]` specifies that an ID is used to indicate a component or attribute. If you omit this parameter, it is assumed that the prompt is used to indicate a component or attribute.

`[-nouserid]` allows you to omit the _user_id: and _password: lines in the REGISTER message.

CAUTION
Risk of compromising workstation security
Do not specify the -nouserid option when setting up the Passport Provisioning API as a network server. If you do, workstation security is compromised because anyone with LAN access will be able to invoke the Passport Provisioning API without validation. Through the use of shell escape, users will also be able to access the UNIX environment with the privileges of the userid that was used to set up the Passport Provisioning API as a network server.

`[-wide <width>]` specifies the maximum length of lines output by the Passport Provisioning API. The default and minimum value is 72. The maximum value is 100000. When you specify values over 80, be careful to avoid any limitations that might exist in the communication path between the Passport Provisioning API and the API User.

`[-echo]` echoes stdin to stdout. This option is intended primarily for improving the readability of the log files that are output when running command files.

`[-trace]` writes debugging information (Protocol Data Units as they are sent and receive tokens as they are parsed, and a few other things) to stderr.

If initialization fails, one or more error messages are output followed by an End message and the Passport Provisioning API terminates. Otherwise, a version message in the following format is output:

```
_version: <version>.<level> Passport Provisioning API
Copyright Nortel Networks 1999
<blank line>
```

Command Filter parameters

For more information on the Provisioning Command Filter API, see 241-6001-209 *Preside MDM Provisioning Command Filter API Reference Guide*.

Use one or more of the following options to invoke the Provisioning Command Filter API with the Passport Provisioning API.

```
/opt/MagellanNMS/bin/pprovapi <provisioning API parameters>
                              [-filter_appl “<UNIX command>”]
                              [-filter_serv <node|IP address> <port>]
                              [-filter_trace] [-filter_width <width>]
```

where:

`<provisioning API parameters>` are one or more of the options described in “Starting the Passport Provisioning API” (page 48).

`-filter_appl` is an option for specifying the 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>` is the UNIX Bourne shell command for invoking the customer application that filters provisioning commands.

`-filter_serv` is an option for specifying 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>` is either the mnemonic or the IP address for a workstation running a Provisioning Command Filter server.

`<port>` is the port of the Provisioning Command Filter server.

`-filter_trace` turns on Provisioning Command Filter API tracing. Trace ouput goes to stderr.

`-filter_width` is an option for specifying the maximum length of lines that are output by the Provisioning Command Filter API Provider. The default is 72.

`<width>` is a number between 72 and 100000.

Running a session

The Passport Provisioning API can only be used to access one Passport switch in one Passport group at a time. The general sequence of messages involved in a session is shown in “Time line diagram” (page 28).

The first request message that you need to send is a REGISTER request message. The REGISTER request message establishes the necessary authorization to use the API on a Passport group. Accessing Passports in a different group can be accomplished in the same API session. However to work on a Passport in a different group, the API User needs to send a DEREGISTER request message to deregister from the existing group and then send a REGISTER request message to register with the new group.

Once you have successfully registered with the Passport switch, you can send request messages to perform provisioning activities and you can continue to do so until you have completed the provisioning work. Then you need to send a DEREGISTER message to deregister from the group, followed by an END request message to terminate the Passport Provisioning API session. For detailed descriptions of the syntax for request messages and the responses to them, see “Writing request messages” (page 60).

There are three modes in which you can use the Passport Provisioning API: interactively, with a command file only, or with a command file and the registration script. Each of these modes is described in the following sections.

Using the API interactively

In the interactive mode, you can start the Passport Provisioning API from a UNIX Access window on the Preside Multiservice Data Manager (MDM) workstation and send it request messages by entering provisioning commands at the keyboard. The interactive mode is useful for performing the following tasks:

- learning about the API, testing potential commands, and reviewing responses to write a successful customer application
- preparing a command file. To test out each command as you build the command file, you can open and edit a new command file in one window, and copy and paste commands to another window that is running the Passport Provisioning API interactively.
- logging in to the MDM workstation from a remote terminal, starting up the Passport Provisioning API on the MDM workstation in interactive mode, and sending commands from the remote terminal to the Passport Provisioning API on the MDM workstation.

To run the Passport Provisioning API interactively from a UNIX Access window on the MDM workstation, you do not need to set up piping because standard input (stdin) and standard output (stdout) captures information from the workstation keyboard and displays output to the screen.

Using the API with a command file

A convenient way to work with the Passport Provisioning API is to create a command file containing request messages and to start up the Passport Provisioning API so it sends the commands contained in the command file.

Using a command file without the register script

Assuming you have created a command file, the syntax of the command you can use to start the Passport Provisioning API is as follows. This command also displays the results on the screen and adds them to a log file.

```
/opt/MagellanNMS/bin/pprovapi -echo < <cmd.file> \
| & tee <log.file>
```

where:

`-echo` echoes input to output so that commands are shown in the output in addition to the responses.

`<cmd.file>` is the absolute path name of the command file to be executed.

`<log.file>` is the absolute pathname of the file in which the results are to be stored.

The first command in the command file must be a REGISTER message, unless you use the register script that is provided with the Passport Provisioning API software. Use of this script is described in "Using a command file with the register script" (page 53).

Example

In this example, the Passport Provisioning API is executed using the command file cmd.tst. The input lines are echoed to stdout:

```
/opt/MagellanNMS/bin/pprovapi -echo < cmd.tst
```

Using a command file with the register script

A register script is provided with the Passport Provisioning API software. This script is the preferred method of starting the Passport Provisioning API to use a command file. The syntax of the command to use the register script is as follows:

```
/opt/MagellanNMS/bin/pregister [-v] [-h] [-id]
```

where:

`[-v]` directs responses for each command to stdout. If this parameter is omitted, the command displays what was uploaded, downloaded, and so on.

`[-h]` displays the syntax of the pregister command.

`[-id]` specifies that the command file uses an ID to identify a component or attribute. If this parameter is omitted, it is assumed that the prompt is used to indicate the component or attribute.

On entering the pregister command, a sequence of prompts is displayed. Respond to these prompts as follows:

1 The first prompt displayed is:

```
Passport Provisioning API command file:
```

Enter the absolute pathname for the command file. After you enter the pathname, the following prompt is displayed:

```
Group name:
```

2 Enter the name of the Passport group that contains the Passport switch on which you are going to perform provisioning. After you enter the group name, the following prompt is displayed:

```
Capability Id:
```

3 Enter the name of the userid that allows you to access the Passport group. After you enter the userid, the following prompt is displayed:

```
Password:
```

4 Enter the password for the group. After you enter the password, the Passport Provisioning API logs in, sends a REGISTER request message, then sends the messages contained in the command file.

Using the API with a propagation log file

When modifying provisioning information (a View) from a Passport switch, you can create a propagation log file and use it as a command file so that the Passport Provisioning API can download the same set of service data changes to the same Passport switch or to a different Passport switch.

> ***Note:*** To apply the propagation log file to a different Passport switch, you need to modify the propagation log file so that all references to the Passport mnemonic name are changed to reflect the new Passport switch to which the propagation log file is being applied.

To create a propagation log file and apply the changes to the same service data view, see “Passport propagate command” (page 155) in 241-6001-023 *Preside MDM Configuration Management for Passport User Guide*.

Terminating Passport Provisioning API Access

Either the API User or the Passport Provisioning API Provider can terminate a provisioning session.

To terminate Passport the Provisioning API access, the API user can enter an end-of-file sequence (control-d) or send the following END request message:

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

The response from the Passport Provisioning API provider is:

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

If the Passport Provisioning API Provider terminates the session, it sends the following response message to the API User:

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

where <termination code data> is one of the following:

Startup failure	The Passport Provisioning API was unable to start up an FFA/FPA process. Since the _version: message has not been output yet, the version is included on the _error: INIT_ERROR line.
PDU failure	The Passport Provisioning API was unable to send or receive a PDU to or from the FFA process. A PDU is a specially formatted IPC message exchanged by provisioning applications. This failure can indicate a software problem.
Lost session	The FFA process aborted the session. This failure can indicate a software problem.
Lost connection	The Passport Provisioning API is unable to communicate with the FFA process. This failure can indicate a software problem.
Out of memory	An attempt to allocate memory failed. This failure indicates that a runaway process is using up memory or that insufficient memory is provisioned on the workstation.

Sample API session

This section contains a sample Passport Provisioning API session that is used to provision data link connection identifier (DLCI) 200 on FrameRelayUni 11. In this sample session, the Passport Provisioning API is invoked interactively without any piping and the method used to specify the component is the use of the prompt name.

1 Start up the Passport Provisioning API

The user starts up the Passport provisioning API by entering the following command:

/opt/MagellanNMS/bin/pprovapi

After initialization, the Passport Provisioning API Provider returns the following message:

```
_version:9.2 Passport Provisioning API Copyright
Nortel Networks 1999
<blank line>
```

2 Register

To authenticate with the Passport group *universe* using the user id *guest* and password *guest*, the API User sends the following REGISTER request message:

```
_cmd: register
_user_id: <Unix userid>
_password: <Unix userid password>
_attr: group S universe
_attr: capabilityid S guest
_attr: password S guest
<blank line>
```

To indicate that the authentication is successful, the Passport Provisioning API Provider returns the following message:

```
<blank line>
_end_resp: REGISTER
_time: 1998 12 05 14 17 21
<blank line>
```

3 Upload the current view

To change the current view for Passport1, the API User sends the following ACTION request message. An upload ACTION message

establishes a session to the Passport and uploads the specified provisioning view.

```
_cmd: ACTION
_obj_id: compid NI EM Passport1
_action_type: upload
_attr: accessmode S UPDATE
_attr: view SS current
<blank line>
```

To indicate that the upload of the current view is successful, the Passport Provisioning API Provider returns the following message:

```
_obj_id: compId NI EM Passport1
_attr: viewname S NMS21.full.984
_attr: sdversion S AC0012c
_attr: swversion S AC0012c
<blank line>
_end_resp: ACTION
_time: 1998 12 05 14 19 13
<blank line>
```

This response message indicates the provisioning view name, the Passport software version (swversion attribute), and other related information. The API User application should verify the Passport version against the version that it expects.

4 Add the new DLCI

To create new DLCI 200 under FrameRelayUni 11, the API User sends the following CREATE request message. A CREATE request message is used to add a new component.

```
_cmd: Create
_obj_id: compid NI FrameRelayUni 11 Dlci 200
_attr: maximumFrameSize I 4096
_attr: committedInformationRate I 12800
<blank line>
```

This REQUEST message also sets the value of the maximumFrameSize to 4096 and the value of the committedInformationRate to 12800. Other attributes are set to their default values.

If the CREATE request is successful, the Passport Provisioning API Provider returns the following message:

```
_obj_class: DataLinkConnectionIdentifier
_obj_id: compId NI FrameRelayUni 11 Dlci 200
<blank line>
_end_resp: CREATE
_time: 1998 12 05 15 38 53
<blank line>
```

5 Perform a semantic check on the modified view

To perform a semantic check on the modified view, the API User sends the following ACTION request message. A Verify ACTION request message is used to request a semantic check.

```
_cmd: ACTION
_obj_class: EM
_obj_id: compID NI EM Passport1
_action_type: verify
<blank line>
```

If the Verify ACTION message is successful, the Passport Provisioning API Provider returns the following message:

```
_obj_class: EM
_obj_id: compID NI EM Passport1
<blank line>
_end_resp: ACTION
_time: 1998 12 05 15 45 53
<blank line>
```

6 Save the modified view

To save the View on the Passport disk using a name determined by the key Preside Multiservice Data Manager (MDM), the API User sends the following ACTION message. A Download ACTION request message saves the provisioning View to the Passport disk using the name specified in the request.

```
_cmd: ACTION
_obj_class: EM
_obj_id: compID NI EM Passport1
_action_type: download
_attr: view SS key NMS
<blank line>
```

If the modified view is successfully saved on the Passport disk, the Passport Provisioning API Provider returns the following message. For this sample session, NMS22.full.990 is used as the name of the view file:

```
_obj_class: EM
_obj_id: compID NI EM Passport1
_attr: viewname S NMS22.full.990
<blank line>
_end_resp: ACTION
_time: 1998 12 05 16 02 00
<blank line>
```

7 The new provisioning View has been saved. You can terminate the API session, work on different Passports in the same group, or continue to work on different Passports in a different group.

- To terminate the API session, the API User sends the following message.

  _end:
  <blank line>

  The Passport Provisioning API Provider returns the following response:

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

- To modify the provisioning data for a different Passport in the same group, the API User sends an upload ACTION message to upload a View file from the new Passport in the authenticated group.
- To modify the provisioning data for a different Passport in a different group, the API User sends the following DEREGISTER request message to turn off authentication for the old group.

  _cmd: DEREGISTER
  <blank line>

  The response message is:

  ```
  <blank line>
  _end_resp: DEREGISTER
  _time: 1994 12 05 16 05 00
  <blank line>
  ```

Now the API User can send a REGISTER request message to authenticate to the new group and continue to work on the Passports in the new group.

Writing request messages

This section contains descriptions of the following items:

- commands that the API User can send in request messages to the Passport Provisioning API Provider
- responses to those commands

Information in this section is intended to assist you in formulating request messages and in understanding the responses to them.

REGISTER requests

The REGISTER message is used to establish the necessary authorization for performing provisioning activities. The first request message in an API session must always be a REGISTER request message.

Request format

The format of the REGISTER message is as follows:

```
_cmd: REGISTER
[_inv_id: <invoke ID>]
_user_id: <Un*x userid>
_password: <Un*x userid password>
_attr: group S <group name>
_attr: capabilityid S <value>
_attr: password S <password>
<blank line>
```

Response format

The format of a response to the REGISTER message is as follows:

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

Note 1: If the -nouserid option has been specified on the command line, the _user_id: and _password: lines are optional.

Note 2: The case (uppercase or lowercase) of the values on the _user_id: and _password: lines is preserved.

Example

An example of a REGISTER request message and the response to it is as follows:

Request:

```
_cmd: REGISTER
_attr: group S noder16_Grp
_attr: capabilityid S systemID
_attr: password S sysPasswd
```

Response:

```
_end_resp: REGISTER
_time: 1995 02 24 08 13 22
```

DEREGISTER requests

The DEREGISTER message is used to discard the authorization established by the last REGISTER command. It is used after a Download or Quit ACTION request in preparation for sending a new REGISTER message.

Request format

The format of the DEREGISTER message is as follows:

```
_cmd: DEREGISTER
[_inv_id: <invoke ID>]
<blank line>
```

Response format

The format of a response to the DEREGISTER message is as follows:

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

Example

An example of a DEREGISTER message and the response to it is as follows:

Request:

```
_cmd: DEREGISTER
```

Response:

```
_end_resp: DEREGISTER
_time: 1995 02 24 08 15 22
```

GET requests

The GET request message is used to retrieve provisioning data. The amount of provisioning data returned can be tailored using the _scope: line.

Request format

The format of the GET message is as follows:

```
_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>
```

where:

`_scope: BASE | NEXT| ALL` determines the provisioning data to be returned, as follows:

- `BASE` returns the attributes for the component named by the _obj_id. This is the default if the _scope: line is not specified.
- `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 sub-hierarchy and their attributes. A separate response message is returned for each object.

`[_attr_id: <attribute name> | ALL]` selects the attribute(s) to be returned. If ALL is specified as an attribute name, all attributes are returned. If no _attr_id: line is specified, only the names of components are returned. More than one _attr_id: line can be specified.

Response format

The format of a response to the GET message is as follows:

```
[_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>
```

Note 1: The values of a LIST (set-valued) attribute are returned in a contiguous set of lines. A line <attribute name> NULL is returned if the list is empty.

Note 2: If the attribute name VECTOR, ARRAY or REPLICATED is specified, each cell value is returned in one line using pseudo-attribute names. This name is constructed by the attribute name and index. For information about vectors, replicated data, and arrays see “Object model” (page 29).

Note 3: To get a particular cell value of the VECTOR, ARRAY or REPLICATED attribute, the pseudo-attribute name must be specified in the GET request.

Example 1

The first example of a GET message and the response to it is as follows:

Request:

```
_cmd: get
_obj_id: compid NI Software NULL
_attr_id: ALL
```

Response:

```
_obj_class: Software
_obj_id: compId NI Software NULL
_attr: avList S “base_AC0323c”
_attr: avList S “networking_AC0323c”
_attr: avList S “trunks_AC0323c”
_attr: avList S “frameRelay_AC0323c”
_attr: avList S “vtds_AC0323c”

_end_resp: GET
_time: 1995 02 24 08 16 28
```

Example 2

The second example of a GET message and the response to it is as follows.

Request:

In this example, the avList is a LIST attribute that contains five items.

```
_cmd: get
_obj_id: compid NI EM NODER16 Vs 401 Framer NULL
_attr_id: ALL
```

Response:

In this example the idleCode and seizeCode are vectors. They are both indexed by A, B, C, and D.

```
_obj_class: Framer
_obj_id: compId NI EM NODER16 Vs 401 Framer NULL
_attr: interfaceName S “EM/NODER16 LogicalProcessor/4
E1/0 Channel/0”
_attr: maxVoiceBitRate S “32”
_attr: minVoiceBitRate S “16”
_attr: maxModemBitRate S “64”
_attr: minModemBitRate S “32”
_attr: audioGain S “-1”
_attr: silenceSuppression S “on”
_attr: echoCancellation S “on”
_attr: aLawConversion S “off”
_attr: transmitBusyYellow S “no”
_attr: transportSignalling S “yes”
_attr: interpretSignalling S “no”
_attr: invertBits S “no”
_attr: signalBits S “ABCD”
_attr: idleCode[S “A”] I 1
_attr: idleCode[S “B”] I 1
_attr: idleCode[S “C”] I 0
_attr: idleCode[S “D”] I 0
_attr: seizeCode[S “A”] I 1
_attr: seizeCode[S “B”] I 1
_attr: seizeCode[S “C”] I 1
_attr: seizeCode[S “D”] I 1

_end_resp: GET
_time: 1995 02 24 08 19 01
```

SET requests

The SET request message is used to change the values of the specified attributes of the indicated provisioning data component.

Request format

The format of the SET request message is as follows:

```
_cmd: SET
[_inv_id: <invoke ID>]
[_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: DEF <attribute name>
_mod: ADD <attribute name> <attribute type> <attribute
value>
_mod: DEL <attribute name> [<attribute type>
<attribute value>]
_mod: REP <attribute name> <attribute type> <attribute
value>
<blank line>
```

where:

`DEF <attribute name>` means clear the LIST or REPLICATED attribute.

`ADD <attribute name>` means add a value to a LIST or REPLICATED attribute. To add a value to REPLICATED data, the attribute name and its associated index (pseudo-attribute name) must be specified.

If the new value already exists in the LIST attribute, it is ignored. If the new cell already exists in the REPLICATED attribute, the value portion of the cell is replaced with the new value.

`DEL <attribute name>` deletes a value from the LIST or REPLICATED attribute. To delete a cell from a REPLICATED attribute, the attribute name and its associated index (pseudo-attribute name) must be specified.

It is considered an error if the DEL modification applies on a non-existing cell in a REPLICATED data or on a non-existing value in a list.

`REP <attribute name>` replaces the attribute value with the specified value.

To replace a value for a VECTOR, ARRAY or REPLICATED attribute, the attribute name and its associated index (pseudo-attribute name) must be specified.

To create a new LIST containing a new set of values, you must specify a DEF or a REP followed by a number of ADDs.

Response format

The format of a response to a SET message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: SET
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example 1

The first example of a SET request message and the response to it is as follows.

Request:

```
_cmd: set
_obj_id: compid NI EM NODER16 Vs 401 Framer NULL
_mod: REP idleCode[S A] I 1
_mod: REP idleCode[S B] I 1
```

Response:

In this example, the vector idleCode's cell A and B are set to 1.

```
_obj_class: Framer
_obj_id: compId NI EM NODER16 Vs 401 Framer NULL

_end_resp: SET
_time: 1995 02 24 08 18 02
```

Example 2

The second example of a SET message and the response to it is as follows.

Request:

```
_cmd: set
_obj_id: compId NI Software NULL
_mod: DEF avList
_mod: ADD avList S base_AC0324a
_mod: ADD avList S networking_AC0324a
_mod: ADD avList S trunks_AC0324a
_mod: ADD avList S frameRelay_AC0324a
_mod: ADD avList S vtds_AC0324a
```

Response:

In this example, the Software avlList is upgraded to version AC0324a.

```
_obj_class: Software
_obj_id: compId NI EM NODER16 Software NULL

_end_resp: SET
_time: 1995 02 24 08 18 02
```

CREATE requests

The CREATE request message is used to establish a new provisioning data component. The _attr: lines provide explicit values for the specified attribute; all other attributes are set to their default values. An error is generated if the component already exists or if its parents are not present.

Request format

The format of a CREATE request message is as follows:

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

Response format

The format of a response to the CREATE request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: CREATE
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

An example of a CREATE request message and the response to it is as follows:

Request:

```
_cmd: create
_obj_id: compid NI FrNni 99
_attr: customerIdentifier I 99
```

Response:

In this example, a new FrNni is added, and the customerIdentifier is set to 99.

```
_obj_class: FrNni
_obj_id: compId NI FrNni 99

_end_resp: CREATE
_time: 1995 02 24 08 22 37
```

Note: Support for mod: lines (see SET request description) has been retained for backwards compatability.

DELETE requests

The DELETE request message is used to remove a specified provisioning data component.

Request format

The format of a DELETE request message is as follows:

```
_cmd: DELETE
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <distinguished name>
<blank line>
```

Response format

The format of a response to the DELETE request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: DELETE
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
<blank line>
```

Example

An example of a DELETE request message and the response to it is as follows:

Request:

```
_cmd: delete
_obj_id: compid NI FrNni 99
```

Response:

```
_obj_class: FrNni
_obj_id: compId NI FrNni 99

_end_resp: DELETE
_time: 1995 02 24 08 23 05
```

ACTION requests

An ACTION request is directed at a specific object and asks that the object perform a specified function. This function can be any of the following:

Upload	establishes a session to the Passport module
Connect	establishes a non-provisioning session to the Passport module
Download	downloads service data to a Passport module
Verify	performs a semantic check on a set of service data for a module
Discard	throws away any changes made to service data since the last upload request message. The discard action message is supported for the uploaded object and on all of its subordinates
Quit	terminates the session. In the case of a provisioning session, the action will discard all changes made to provisioning data since the last upload message. The quit action message is supported for the component in the most recent upload action
Ascii_cmd	allows the user to send ASCII (telnet-like) command to the Passport node via the FMIP interface

Upload ACTION requests

The Upload ACTION request message is used to establish a session to the Passport module and specify access to a specific view of provisioning data for a specific Passport. This action request is required before provisioning data can be examined or modified.

If another provisioning data view is required, a Quit or Download ACTION request must be sent and the Upload ACTION request re-issued with the new parameters.

Request format

The format of an Upload ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <2|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>
```

Note 1: The default value for the preload attribute is YES.

Note 2: The default value for the accessmode attribute is READ.

Note 3: The default value for view is EDIT. The editing view is uploaded.

Note 4: The case of view filename is preserved.

Note 5: Specify 2 if component attribute identifiers (IDs) are being used to identify components. Otherwise, specify EM.

Response format

The format of a response to the Upload ACTION request is as follows.

```
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <2|EM> <Passport name>
[_attr: view S <view name>]
_attr: sdversion S <SD version>
_attr: swversion S <Passport version>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

where:

`[yes|no]` specifies whether the provisioning stack preloads the service data view into its internal instance hierarchy from the Passport module when a client accesses the view. The default is yes. It may be useful to specify no to save time when only small changes have been made to a large service data view.

`<2|EM>` is either a component identifier (2) if IDs are being used to identify components, or a prompt or full name (EM) if prompts or full names are being used to identify components.

`<view name>` is the view name.

`<SD version>` is the version of the activation file that is used by the provisioning stack.

CAUTION

Risk of difficulty in provisioning <SD version> does not match the <Passport version>

The version of the activation file (SD version) must match the version of software in the Passport (Passport version). If they do not, you will experience difficulty performing provisioning operations with the API. If they do not match, either send a Quit ACTION request to end the upload, or ensure that you are using object model files that were generated from an activation file whose version is compatible with the version of the Passport software.

`<Passport version>` is the version of software that is loaded into the Passport switch.

Example

An example of an Upload ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_class: EM
_obj_id: compid NI EM NODER16
```

```
_action_type: upload
_attr: accessmode S UPDATE
_attr: view SS current
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16
_attr: viewname S NMS22.full.998
_attr: sdversion S AC0323c
_attr: swversion S AC0323c

_end_resp: ACTION
_time: 1995 02 24 08 15 05
```

Connect ACTION requests

The Connect ACTION request message is used to establish a non-provisioning session to the Passport module in order to access the operational data of a specific Passport. This action request is required before operational data can be examined or modified.

If another provisioning or non-provisioning session is required, a Quit ACTION request must be sent, and the Connect or Upload ACTION request issued with the new parameters.

Request format

The format of an Connect ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
_action_type: connect
<blank line>
```

Response format

The format of a response to the Upload ACTION request is as follows.

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

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

where:

`<SD version>` is the version of the activation file that is used by the provisioning stack.

CAUTION

Risk of difficulty in provisioning <SD version> does not match the <Passport version>

The version of the activation file (SD version) must match the version of software in the Passport (Passport version). If they do not, you will experience difficulty performing provisioning operations with the API. If they do not match, either send a Quit ACTION request to end the upload, or ensure that you are using object model files that were generated from an activation file whose version is compatible with the version of the Passport software.

`<Passport version>` is the version of software that is loaded into the Passport switch.

Example

An example of an Connect ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_class: EM
_obj_id: compid NI EM NODER16
_action_type: connect
_attr: accessmode S READ
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16
_attr: sdversion S AC0323c
_attr: swversion S AC0323c
```

```
_end_resp: ACTION
_time: 1997 02 24 08 15 05
```

Download ACTION requests

The Download ACTION request message downloads provisioning data to a Passport module and closes the session. This request does not apply to the non-provisioning session.

Request format

The format of a Download ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <2|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>
```

Note 1: The default value for the check attribute is CHANGE.

Note 2: If the NOCHECK option is specified, the view is saved on the Passport disk without performing the semantic checks.

Note 3: If the CHANGE option is specified, only the modified components are checked. If the Passport does not support this option, a full check is performed.

Note 4: If the COMPLETE option is specified, all components are checked.

Note 5: The default value for the *stoponerr* attribute is YES. The check should be aborted immediately upon finding the first error. If the Passport does not support this option, all errors are reported.

Note 6: If the save ASCII option is specified, the view is saved in an ascii format.

Note 7: If the save PORTABLE option is specified, the view is saved in an portable format.

Note 8: Two options ASCII and PORTABLE can both be specified. If they are, two provisioning files are saved: one in PORTABLE format and the other in ASCII format.

Note 9: If the save option is not specified, the Passport switch decides how to save the view.

Response format

The format of a response to the Download ACTION request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <2|EM> <Passport name>
_attr: viewname S <view name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

An example of a Download ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_id: compid NI EM NODER16
_action_type: download
_attr: view SS viewname newname
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16
_attr: viewname S newName.full.001

_end_resp: ACTION
_time: 1995 02 24 09 56 26
```

Quit ACTION

The Quit ACTION request message terminates the provisioning or non-provisioning session. In the case of a provisioning session, provisioning data changes that were made since the most recent upload are lost.

Request format

The format of a Quit ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <2|EM> <Passport name>
_action_type: quit
<blank line>
```

Response format

The response to a Quit ACTION request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <2|EM> <Passport name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

An example of a Quit ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_id: compid NI EM NODER16
_action_type: quit
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16

_end_resp: ACTION
_time: 1995 02 24 08 35 01
```

Discard ACTION requests

The Discard ACTION request message discards any changes made to the specified component (and any subcomponents) since the last Upload ACTION request message. This request does not apply to a non-provisioning session.

Request format

The format of a Discard ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <distinguished name>
_action_type: discard
<blank line>
```

Response format

The format of a Discard ACTION request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

An example of a Discard ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_id: compid NI EM NODER16 Software
_action_type: discard
```

Response:

```
_obj_class: Software
_obj_id: compId NI EM NODER16 Software

_end_resp: ACTION
_time: 1995 02 24 08 29 12
```

Verify ACTION requests

The Verify ACTION request message is used to perform semantic checks on the specified component (and on its subordinates). This request does not apply to a non-provisioning session.

Request format

The format of a Verify ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI <distinguished name>
_action_type: verify
<blank line>
```

Response format

The format of a response to the Verify ACTION request message is as follows:

```
[_inv_id: <invoke ID>]
_obj_class: <object class>
_obj_id: compId NI <distinguished name>
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

An example of a Verify ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_id: compid NI EM NODER16
_action_type: verify
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16

_end_resp: ACTION
_time: 1995 02 24 08 24 01
```

Ascii_cmd ACTION requests

The Ascii_cmd ACTION request message is used to allow the user to send ASCII (telnet-like) command to the Passport node via the FMIP interface.

Request format

The format of an Ascii_cmd ACTION request message is as follows:

```
_cmd: ACTION
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
_action_type: ascii_cmd
_attr: command SS <ascii command string>
<blank line>
```

Response format

The format of a response to the Ascii_cmd ACTION request is as follows.

```
[_inv_id: <invoke ID>]
[_obj_class: <object class>]
_obj_id: compId NI EM <Passport name>
_attr: response S <response>
[_attr: response S <response>]
...
<blank line>
[_inv_id: <invoke ID>]
_end_resp: ACTION
_time: <year> <month> <day> <hour> <minute> <second>
<blank line>
```

Example

An example of an Ascii_cmd ACTION request message and the response to it is as follows:

Request:

```
_cmd: ACTION
_obj_class: EM
_obj_id: compid NI EM NODER16
_action_type: ascii_cmd
_attr: command SS l shelf
```

Response:

```
_obj_class: EM
_obj_id: compId NI EM NODER16
_attr: response S “shelf card/0”
_attr: response S “shelf card/1”
...
```

```
_end_resp: ACTION
_time: 1997 02 24 08 15 05
```

Chapter 4
API model difference tool

This section describes the tool you can use to identify the differences between two Provisioning API model files. See the following sections for more information:

About the API model difference tool

The API model difference tool is a UNIX line command that compares two Provisioning API model files and reports any differences. By default, this tool reports on all templates, constructs, service data components, and service data attributes in the model files. If you do not need to know everything that has changed, you can restrict the report to templates, constructs, service data components, and service data attributes that you specify.

Model files

You can regenerate the required Provisioning API model files by using the ppmod command. You can include operational components in the Provisioning API data model files by specifying the -c option when using the ppmod command. Or, if you are using ppmod interactively, respond with y to the prompt asking if you want operational components included. The resulting model files have the _all suffix in their names, which indicates they include operational components. See "Generating a Passport Provisioning API Model" (page 43) for more information on ppmod.

The Passport activation file, which is used as input to ppmod, may not be available. It it is not available, you can get it by using the fmsgetmod command. See the 241-6001-303 *Preside MDM Administrator Guide* for more information on fmsgetmod.

The ppmod command creates three types of model files: comps, attrs, and hier. The difference tool only compares two comps (or comps_all) files or two attrs (or attrs_all) files; it cannot compare two hier files.

Command line syntax

The format for the difference tool command is:

```
/opt/MagellanNMS/bin/ppapidiff [-h | -help]
                                or
   [-components <components listfile>]
   [-attrbutes <attribute list file>]
   [-templates <templates list file>]
   [-constructs <constructs list file>]
   [-debug] [-trace] [-all] <old stem> <new stem>
```

where:

`-h | -help` displays command line usage.

`-components` specifies a file that lists the component names for which changes are to be reported. The file contains one component name to a line. If you do not use this option, the difference tool reports changes for all components. You can specify more than one file by repeating this option.

`-attributes` specifies a file that lists the attribute names for which changes are to be reported. The file contains one attribute name to a line. If you do not use this option, the difference tool reports changes for all attributes. You can specify more than one file by repeating this option.

`-templates` specifies a file that lists the template names for which changes are to be reported. The file contains one template name to a line. If you do not use this option, the difference tool reports changes for all templates. You can specify more than one file by repeating this option.

`-constructs` specifies a file that lists the construct names for which changes are to be reported. The file contains one construct name to a line. If you do not use this option, the difference tool reports changes for all constructs. You can specify more than one file by repeating this option.

> ***Note:*** The ID, ATTRIBUTE ID, and IS MEMBER OF constructs are always displayed because they are part of the identification of a component or attribute.

`-debug` writes high level debugging information to stderr.

`-trace` writes detailed low level information to stderr. This option generates a great deal of output.

`-all` uses the Provisioning API model files with the _all suffix.

`old stem` specifies the initial part of the old Provisioning API model file; for example, /opt/MagellanNMS/cfg/PassportSchema/BC03E. The .comps and .attrs or the .comps_all and .attrs_all suffixes are added by the difference tool as needed in order to form filenames.

`new stem` specifies the initial part of the new Provisioning API model file.

You must always list the old file before the new file. Compressed files (suffixes .Z, .z, or .gz) can be used for either the old file or the new file. The difference tool looks for and processes compressed files if the model files cannot be found in their regular form.

The output from the difference tool is written to stdout. You can use standard redirection to write the output to a file. Informational messages are written to stdout. Error messages are written to stderr and the difference tool terminates.

Example

`/opt/MagellanNMS/bin/ppapidiff BC03E BD0028B`

A sample of the response is:

```
Changed Passport API Data Model file: BC03E.comps
                                  to: BD0028B.comps
...
Changed: OBJECT CLASS VirtualPathConnection
```

```
            ID: 10230
            IS MEMBER OF: EM/AtmIf
      Changed: *Key value
          Deleted: 1..255
            Added: 0..255

Changed: OBJECT CLASS VirtualChannelConnection
            ID: 10137
            IS MEMBER OF: EM/AtmIf
      Changed: MAY CONTAIN
          Added: 10200 (Src) SourcePvc      DELETE CREATE GET
...
Changed Passport API Data Model file: BC03E.attrs
                                    to: BD0028B.attrs
...
Changed: ATTRIBUTE lineType
            ID: 520
            IS MEMBER OF: EM/Lp/E1
      Changed: LEGAL VALUES
           Added: unframed
           Added: basicframe
           Added: multiframe

Changed: ATTRIBUTE clockingSource
            ID: 521
            IS MEMBER OF: EM/Lp/E1
      Changed: LEGAL VALUES
           Added: otherPort
           Added: srtsMode
       Added: DEFAULT: “line”

Changed: ATTRIBUTE overrideTransmitSpeed
            ID: 636
      Changed SYNTAX: L
                  to: I
      Changed: LEGAL VALUES
         Deleted: 0..4294967295
           Added: 0
           Added: 1000..2147483647

Changed: ATTRIBUTE maximumErroredInterval
            ID: 648
```

```
   Changed: IS MEMBER OF
      Deleted: EM/Trk/Utp
...
```

Error messages

The table “Error messages for difference tool” (page 87) lists the error messages that the difference tool can generate. The $<n> items indicate values that are filled in at run time.

Table 1
Error messages for difference tool

Message and Action
`ERROR: Invalid option: $0` The command line parameters are incorrectly specified. Correct and retry.
`ERROR: More than two file stems specified.` The model files are incorrectly specified. Correct and retry.
`ERROR: Less than two file stems specified.` The model files are incorrectly specified. Correct and retry.
`ERROR: Missing filename for -components option.` Correct and retry.
`ERROR: Missing filename for -attributes option.` Correct and retry.
`ERROR: Missing filename for -templates option.` Correct and retry.
`ERROR: Missing filename for -constructs option.` Correct and retry.
`ERROR: Unable to open $0: $1` The indicated model file, components list file, attributes list file, templates list file, or constructs list file ($0) cannot be opened. The UNIX error message ($1) should provide the information to determine the cause of the problem and how to correct it.
(Sheet 1 of 4)

Table 1 (continued)
Error messages for difference tool

Message and Action
`ERROR: Invalid template header line in $0 ($1): $2` The first line in a template does not have the expected format. The filename ($0), line number ($1), and text of the line in error ($2) are provided. Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current Preside Multiservice Data Manager (MDM) software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
`ERROR: Invalid construct line in $0 ($1): $2` A construct line does not have the expected format. The filename ($0), line number ($1), and text of the line in error ($2) are provided. Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current MDM software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
`ERROR: Missing '}' in $0 ($1): $2` A set of values, started by an opening brace bracket ({) is not terminated by a closing brace bracket (}). The filename ($0), line number ($1), and text of the line in error ($2) are provided. Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current MDM software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
(Sheet 2 of 4)

Table 1 (continued)
Error messages for difference tool

Message and Action
`ERROR: Invalid line in $0 ($1): $2` Indeterminate parsing error. The filename ($0), line number ($1), and text of the line in error ($2) are provided. Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current MDM software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
`ERROR: Missing terminating quote in $0 ($1): $2` The terminating quote for a quoted string is missing, or there are illegal quotes in the line. The filename ($0), line number ($1), and text of the line in error ($2) are provided. Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current MDM software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
`ERROR: Premature EOF on $0` Verify that the correct model files are specified on the command line. Regenerate the model files if they have not been generated at the current MDM software level. If there is an obvious error in the model file, you can manually edit the file. However you still need to report the problem to the System Administrator.
(Sheet 3 of 4)

Table 1 (continued)
Error messages for difference tool

Message and Action
`ERROR: Insufficient memory` The workstation may be overloaded or misconfigured. Another workstation may be able to run the tool, or the same workstation may be able to run the tool at another time. Contact the System Administrator regarding the configuration of the workstation.
`ERROR: $0: $1` $0 is a filename and $1 is a UNIX error message that indicates a problem with an attempt to open the file. Use the information in the UNIX error message to correct the problem.
(Sheet 4 of 4)

Appendix A
Compliance statement

This appendix describes how the Passport Provisioning API complies with information in the API Primer. It lists the various API Primer features and indicates which features are supported. In addition, the statement defines the constraints, limitations, and conditions for the support of these features.

The tables in this appendix contain the following information:

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

Each box in the Supported column of the tables is filled in 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. The conditions are described in notes at the end of the table.
- N/A, not applicable

See the following sections for more information:

- “API message types” (page 92)
- “API message lines” (page 93)

- “API data types” (page 98)
- “Sieve object support” (page 99)

API message types

The table “API message types” (page 92) indicates which messages are supported by the Passport Provisioning API. If a message is in this table, it means that the message can be generated or received with all of the mandatary lines. The Direction column indicates whether the message is sent or received by the Passport Provisioning API Provider.

Table 2
API message types

Message type	Direction	Supported
GET request	Receive	Y
GET response	Send	Y
SET request	Receive	Y
SET response	Send	Y
ACTION request	Receive	Y
ACTION response	Send	Y
CREATE request	Receive	Y
CREATE response	Send	Y
DELETE request	Receive	Y
DELETE response	Send	Y
REGISTER request	Receive	Y
REGISTER response	Send	Y
DEREGISTER request	Receive	Y
DEREGISTER response	Send	Y
EVENT-REPORT message	Send	N
END OF RESPONSE message	Send	Y
ERROR message	Send	Y

(Sheet 1 of 2)

Table 2 (continued)
API message types

Message type	Direction	Supported
block construct	Send	N
VERSION message	Send	Y
END message	Send	Y
END message	Receive	Y
echo message	Receive	Y
trace message	Receive	Y

(Sheet 2 of 2)

API message lines

The table “API message lines” (page 93) indicates which message lines and keywords are supported by the Provisioning API. It describes overall support for the message line .The table “Optional message lines” (page 96) gives more information on a per-message basis. Message lines that are mandatory are listed in “Optional message lines” (page 96), 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, Connect, and Ascii_cmd	Y
_attr		Y
_attr_id	general support ALL	Y Y
_block		N
_capability		N

(Sheet 1 of 4)

Table 3 (continued)
API message lines

Message line	Keywords	Supported
_cmd	REGISTER, DEREGISTER, ACTION, CREATE, SET, DELETE, and GET	Y (see also “API message types” (page 92))
_echo	ON, OFF	Y
_end	general support	Y
	USER_REQUEST	Y
	FATAL_ERROR	Y
	SYSTEM_SHUTDOWN	N
	LOST_CONNECTION	N
	LOST_SESSION	N
_end_block		N
_error	general support	Y
	ACCESS_DENIED	N
	APPLICATION_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_CLASS	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

(Sheet 2 of 4)

Table 3 (continued)
API message lines

Message line	Keywords	Supported
_filter	general support P EQ NE LT GT LE GE LEFT MIDDLE RIGHT IN NOTIN	N
_inv_id		Y
_mod	general support ADD DEL DEF REP	Y (See note 1) N (See note 1) N (See note 1) N (See note 1) Y (See note 1)
_obj_id		Y
_obj_class		Y
_password		Y
_ref_obj_id		N
_scope	general support BASE NEXT ALL	C (See note 2) C (See note 2) C (See note 2) C (See note 2)
_sieve_id		N
_sup_obj_id		N
_time		Y
_trace	ON, OFF	Y
_user_id		Y

(Sheet 3 of 4)

Table 3 (continued)
API message lines

Message line	Keywords	Supported
_version		Y
Note 1: In the SET command, supports REP for all attributes, but ADD DEL or DEF will cause an error if it cannot be applied to an attribute. ***Note 2:*** The GET command supports scopes of BASE, NEXT, and ALL. The DELETE command supports a scope of BASE only.		
(Sheet 4 of 4)		

Table 4
Optional message lines

Message type	Message line	Supported
GET request	_inv_id	Y
	_scope	Y
	_filter	N
	_attr_id	Y
GET response	_inv_id	Y
	_attr	Y
SET request	_inv_id	Y
	_scope	C (See note 1)
	_filter	N
	_mod	C (See note 2 and note 3)
SET response	_inv_id	Y
	_attr	N
ACTION request	_inv_id	Y
	_scope	N
	_filter	N
	_attr	C (See note 4)
ACTION response	_inv_id	Y
	_attr	C (See note 4)
(Sheet 1 of 3)		

Table 4 (continued)
Optional message lines

Message type	Message line	Supported
CREATE request	_inv_id	Y
	_obj_id	Y
	_sup_obj_id	N
	_ref_obj_id	N
	_attr	Y
CREATE response	_inv_id	Y
DELETE request	_inv_id	Y
	_scope	C (See note 5)
	_filter	N
DELETE response	_inv_id	Y
REGISTER request	_inv_id	Y
	_user_id	Y
	_password	Y
	_attr	Y
REGISTER response	_inv_id	Y
	_user_id	N
	_capability	N
	_attr	N
DEREGISTER request	_inv_id	Y
DEREGISTER response	_inv_id	Y
event-report message	_attr	N
end-of-response message	_inv_id	Y
error message	_inv_id	Y
	_attr	Y

Note 1: The SET request message only supports a scope of BASE.

Note 2: The SET request message only supports modifications (_mod lines) of DEF, ADD or DEL for LIST or REPLICATED data attributes.

Note 3: A SET request message must include at least one *_mod* line.

(Sheet 2 of 3)

Table 4 (continued)
Optional message lines

Message type	Message line	Supported
Note 4: There are _attributes lines in the Upload, Download, and Ascii_cmd ACTION requests, but not in Connect, Quit, Discard, and Verify ACTION requests.		
Note 5: The DELETE command only supports a scope of ALL.		
(Sheet 3 of 3)		

API data types

The table "API data types" (page 98) indicates the API data types supported by the Passport Provisioning 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)	N
D (Date/time)	C (See note)
E (Enumerated)	C (See note)
FS (Formatted String)	Y
I (Integer)	Y
H (Hexadecimal)	C (See note)
L (Long Integer)	Y
LI (Link Identifier)	N
NI (Node Identifier)	Y
RD (Range of Data)	N
RI (Range of Integers)	N
RS (Range of Strings)	N
S (String)	Y
(Sheet 1 of 2)	

Table 5 (continued)
API data types

API data type	Supported
SB (Sequence of Booleans)	N
SI (Sequence of Integers)	Y
SS (Sequence of Strings)	Y
Note: It is considered as a string (S).	
(Sheet 2 of 2)	

Sieve object support

The table "Sieve object attributes" (page 99) indicates which attributes of the sieve object are supported by the Provisioning API.

Table 6
Sieve object attributes

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

Appendix B
Error messages

This appendix provides a list of error messages for the Passport Provisioning API. See the following sections for more information:

- "Responses to undesired events" (page 101)
- "Error messages" (page 103)

Responses to undesired events

When an error occurs, one or more error messages are returned. If the error is not recoverable, that is it cannot be corrected by issuing a corrected request message, an _end line is also returned and the Passport Provisioning API terminates.

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 API:

`INT_ERROR`

occurs during Passport Provisioning API Provider initialization. The version number is included on the _error: line. Since the version message is not output (because 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.

`NOT_SUPPORTED`

occurs when the specified element, although part of the overall API grammar, is not supported by the Provisioning API.

`OUT_OF_SEQUENCE`

occurs when the request is not valid within the current context (for example, an Upload ACTION request before a REGISTER request, or a GET or SET request before a successful Upload ACTION request).

`APPLICATION_ERROR`

indicates that the request was syntactically correct, but semantically incorrect and the provisioning stack has returned an error Protocol Data Unit (PDU) in response to the request.

If an error PDU is returned from the provisioning stack, 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 Passport Provisioning API Provider returns an error message, followed by an end message, then it terminates.

Error messages

The table “Error messages” (page 103) lists the error messages that can be generated by the Provisioning API. The items in the table listed as $<n> indicate values that are filled in at runtime. For completeness, we have listed these items in the table “Phrases that can substituted for $<n> variables in the error messages in the table “Error messages” (page 103)” (page 111).

Table 7
Error messages

Code	Description and possible solution
A0001	`Unknown session status received: $0`
	Report this internal error to your System Administrator.
A0002	`Unknown transaction status received: $0`
	Report this internal error to your System Administrator.
A0003	`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.
A0004	`PDU failure.`
	Report this internal error to your System Administrator.
A0005	`Lost connection.`
	This could be due to a software failure in one of the servers, in which case it should be reported to your System Administrator. It may also be due to a connectivity problem between the processes that need to be involved in running this application. Check with your local UNIX administrator to correct any local network problems and/or UNIX software problems and retry.
A0007	`Out of memory.`
	Either there is not enough real memory and/or swap space and these need to be increased, or the workstation is overloaded and the workload needs to be reduced or rescheduled.

(Sheet 1 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
A0008	`Improperly terminated string.` A quoted string does not end with a quote or it does not end with the same type of quote (single or double) with which it began. Make sure that quoted strings end with a quote and that they are of the same type of quote (single or double) with which the string began.
A0009	`Unknown connection status received: $0` Report this internal error to your System Administrator.
A0010	`New transaction on session.` Report this internal error to your System Administrator.
A0011	`New session on connection.` Report this internal error to your System Administrator.
A0012	`Invalid option: $0` Check the command line that was entered for spelling mistakes. Reread the descriptions of the command line parameters. Re-enter a corrected command line.
A0013	`Unable to startup $0 with args $1 $2: $3` UNIX was not able to start up the server (value substituted for $0).
A0014	`Unable to initialize PROP PDO/PDU.` Report this internal software failure to your System Administrator.
A0017	`Missing width.` The -width option was specified without a width. Re-enter the command this time remembering to specify the width for the *-width* option.
A0018	`Invalid width: $0` The width specified is outside the range 72 to 100000, or it is non-numeric. Re-enter the command with a numeric width that is within the range.
A0019	`Unable to connect to server.` The server failed shortly after startup. Check the log file and/or console for error messages to determine the cause of the failure.

(Sheet 2 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
A0020	`Unable to determine login userid.` Something is wrong with the UNIX environment. Check with the UNIX system administrator to determine the cause and solution.
A0101	`Missing_action_type: line` An ACTION request message does not have an _action_type: line. Re-enter the action request message, this time specifying the desired action in an _action_type line.
A0109	`Missing $0.` A syntactical element is missing from a line. The phrase substituted for $0 specifies the element that is missing. Re-enter with the element filled in.
A0110	`Invalid $0: $1` A syntactical element has not been properly formed. A phrase describing the element is substituted for $0. The actual token that is in error is substituted for $1.
A0113	`Unsupported $0: $1` The syntactical element described by the phrase substituted for $0 is not supported by the Passport Provisioning API. The token that is not being supported is substituted for $1. Correct and retry.
A0114	`Extraneous data at end of line: $0` There is extra data that is not required at the end of the line. Either the data is truly superfluous and should be removed, or there is an error earlier in the line that needs to be corrected (For example, the wrong data type was specified).
A0202	`Invalid component id.` An odd number of tokens was specified for a component id. A component id consists of category and key value pairs and must, therefore, have an even number of tokens. Correct and re-enter.
A0301	`REGISTER request is not valid in current context.` The REGISTER request is valid only at startup or after a successful DEREGISTER request (which, in turn, is valid only after a successful Download ACTION request or Quit ACTION request). Correct and retry.

(Sheet 3 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
A0302	`Invalid UNIX userid/password specification.` The UNIX userid and password specified in the REGISTER request were not validated by UNIX. Use a correct UNIX userid and password combination. If the Passport Provisioning API Provider is not being used as a network server, it is reasonable to specify the -nouserid option and omit the _user_id: and _password: lines from REGISTER requests.
A0401	`Upload action is not valid in the current context.` The Upload ACTION is valid only after a successful REGISTER request, Download ACTION request, or Quit ACTION request. Correct and retry.
A0501	`GET request is not valid in the current context.` GET requests are valid only after a successful Upload ACTION request until a successful Download or Quit ACTION request. Correct and retry.
A0502	`Requested object(s) not found.` This is most likely the result of naming a service data field as if it were an object rather than an attribute. If the component does not really exist, a different error message with a different error message code from *server* is presented. Try removing the last category and key value pair from the _obj_id: line. If all attributes are not being requested, put the last category in an _attr_id: line.
A0601	`SET request is not valid in the current context.` SET requests are valid only after a successful Upload ACTION request until a successful Download ACTION or Quit ACTION request. Correct and retry.
A0602	`Modifications missing from SET request.` At least one modification must be specified in a SET request message. Correct and retry.
A0603	`Only BASE scope is supported for the SET request.` Either remove the _scope: line or specify a scope of BASE.
A0701	`CREATE request is not valid in the current context.` CREATE requests are valid only after a successful Upload ACTION request until a successful Download or Quit ACTION request. Correct and retry

(Sheet 4 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
A0801	`DELETE request is not valid in the current context.`
	DELETE requests are valid only after a successful Upload ACTION request until a successful Download or Quit ACTION request. Correct and retry.
A0901	`Download action is not valid in the current context.`
	Download ACTION requests are valid only after a successful Upload ACTION request until a successful Download or Quit ACTION request. Correct and retry.
A1001	`Quit ACTION is not valid in the current context.`
	The Quit ACTION request is valid only after a successful Upload ACTION request and until a successful Quit or Download ACTION request. Correct and retry
A1101	`Missing object id.`
	Many API requests operate on a specific service data component and that component needs to be explicitly specified in an _obj_id: line. Correct and retry.
A1102	`Unexpected PDU reply code: $0`
	Report this internal software error to your System Administrator.
A1201	`Discard action request is not valid in the current context.`
	The Discard ACTION request is valid only after a successful Upload ACTION request and until a successful Quit or Download ACTION request. Correct and retry.
A3101	`DEREGISTER request is not valid in current context.`
	The DEREGISTER request is valid only after a successful REGISTER request, or a successful Quit or Download ACTION request. Correct and retry.
A1401	`Verify ACTION request is not valid in the current context.`
	The Verify ACTION request is valid only after a successful Upload ACTION request and until a successful Quit or Download ACTION request. Correct and retry.
P0010	`Can not apply the DEF modification for a structure attribute cell.`
	Correct and retry.
P0011	Array cell modification must be REP.
	Correct and retry.

(Sheet 5 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
P0012	Attribute $0 does not have the same number of index.
	Correct and retry.In the error message, the attribute name is substituted for $0.
P0013	Attribute $0: modification $1 conflicts with other modification(s)
	Correct and retry. In the error message, the attribute name is substituted for $0 and the invalid modification description is substituted for $1.
P0014	Object id: $0 is required for ascii_cmd ACTION
	Correct and retry. In the error message EM or 2 is substituted for $0
P0015	Object id: $0 is required for connect ACTION
	Correct and retry. In the error message EM or 2 is substituted for $0
P0016	ID required for $0: $1:
	The Passport Provisioning API has been started up using the -id option. You must specify an ID, not a prompt or an abbreviation name. Correct and retry. In the error message, the object class is substituted for $0 and the object class that was specified in the request is substituted for $1.
P0017	Object id: $0 is required for upload ACTION
	Correct and retry. In the error message EM or 2 is substituted for $0.
P0018	Object id: $0 is required for download ACTION.
	Correct and retry. In the error message EM or 2 is substituted for $0.
P0019	Object id: $0 is required for quit ACTION.
	Correct and retry. In the error message EM or 2 is substituted for $0.
P0020	Invalid index: Missing [.
	Correct and retry.
P0021	Invalid index: Missing].
	Correct and retry.
P0022	Invalid index: unexpected].
	Correct and retry

(Sheet 6 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
P0023	Invalid index: unexpected [.
	Correct and retry
P0025	Attribute $0: Extraneous data ($1).
	Correct and retry. In the error message, the attribute name is substituted for $0 and the extraneous data is substituted for $1.
P0026	Attribute $0: Missing index type.
	Correct and retry. In the error message, the attribute name is substituted for $0.
P0027	Attribute $0: Missing index value.
	Correct and retry. In the error message, the attribute name is substituted for $0.
P0029	Attribute $0: Invalid index specified.
	Correct and retry. In the error message, the attribute name is substituted for $0.
P0030	Invalid attribute pseudo-name ($0) specified.
	Correct and retry. In the error message the pseudo-name is substituted for $0.
P0031	Invalid attribute pseudo-name specified.
	Correct and retry.
P0303	`Incomplete Passport Group access specification.`
	All three Passport Group access attributes must be specified. Correct and retry.
P0402	`Unable to extract View file name from Open Session reply.`
	Report this internal software error to your System Administrator.
P0403	`(Unable to extract Activation file version from Open Session reply.`
	Report this internal software error to your System Administrator.
P0404	`Unable to extract Passport version from Open Session Reply.`
	Report this internal software error to your System Administrator.
P0500	Provisioning view is opened with READ only access mode.
	Send an upload ACTION request message that contains an UPDATE option.

(Sheet 7 of 8)

Table 7 (continued)
Error messages

Code	Description and possible solution
P0501	Provisioning view has not been uploaded.
	Send an upload ACTION request message to upload the view.
P0502	Registration has not been performed.
	Send a REGISTER request message to register with the Passport group.
P0503	Provisioning view has already been uploaded.
	You have already uploaded the specified view.
P0505	Registration is already done.
	You have already registered with the Passport group.
P0506	Connect request is not valid in current context
	The connect ACTION is valid only after a successful REGISTER request. Correct and retry.
P0508	Ascii_cmd request is not valid in current context
	The ascii_cmd ACTION is valid only after a successful connect ACTION request. Correct and retry.
P0902	Unable to extract the View name from the Download reply
	Report this internal software error to your System Administrator.
P0903	`CURRENT view cannot be downloaded.`
	Specify the full view name or the key view on downloads.
P0904	`COMMITTED view can not be downloaded.`
	Specify the full view name or key view on downloads.
P0905	`EDIT view can not be downloaded.`
	Specify the full view name or key view on downloads.

(Sheet 8 of 8)

Table 8
Phrases that can be substituted for $<n> variables in the error messages in the table "Error messages" (page 103)

Code	Phrase
A0151	label of first line in request message
A0152	request code
A0153	end message line label
A0154	ACTION message line label
A0155	ACTION type
A0156	object ID keyword
A0157	object ID type
A0158	invoke ID
A0159	object class
A0164	key
A0165	view type
A0166	Service Data view type
A0169	scope
A0170	attribute name
A0171	attribute value type
A0172	attribute value
A0173	date key
A0357	REGISTER request attribute
A0358	UNIX password
A0359	UNIX userid
A0360	REGISTER message line label
A0453	access mode
A0454	access mode type
A0455	Upload ACTION attribute

(Sheet 1 of 2)

Table 8 (continued)
Phrases that can be substituted for $<n> variables in the error messages in the table "Error messages" (page 103)

Code	Phrase
A0456	Upload ACTION message line label
A0551	attribute name
A0552	GET message line label
A0651	modification type
A0652	SET message line label
A0751	CREATE message line label
A0851	DELETE message line label
A0953	Download ACTION attribute
A0954	Download ACTION message line label
A0956	download check type
A1051	Quit ACTION message line label
A1251	Discard ACTION message line label
A1351	DEREGISTER message line label
A1451	Verify ACTION message line label
P0159	compid NI object class
P0163	view name
P0351	password attribute value
P0352	attribute type for password
P0353	capability attribute value
P0354	attribute type for capability
P0355	group attribute value
P0507	connect ACTION request line label
P0509	ascii_cmd ACTION request line label

(Sheet 2 of 2)

Index

O

P

R

S

Preside Multiservice Data Manager

Passport Provisioning API

Reference Guide

Release: R14.3

Publication: 241-6001-207
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003
Printed in Canada

NORTEL NETWORKS