NORTEL NETWORKS

Passport 7400, 15000, 20000

SNMP Guide

241-5701-300

Passport 7400, 15000, 20000

SNMP Guide

Publication: 241-5701-300
Document status: Standard
Document version: 5.2S1
Document date: November 2003

Printed in Canada

Publication history

November 2003

5.2S1 Standard
General availability. Contains information on Passport 7400, Passport 15000, and Passport 20000 for the PCR5.2 GA release.

Contents

List of figures

List of tables

About this document

This document describes how the Simple Network Management Protocol (SNMP) can be used to manage Passport nodes.

The following topics are discussed in this section:

Who should read this document and why

This document is intended for those who install, provision, use, and troubleshoot SNMP on Passport systems.

What you need to know

Before you read this document, you should be familiar with basic data communication concepts and terms.

Passport 7400 users should also read 241-5701-030 *Passport 7400, 15000, 20000 Overview*.

Passport 15000 and 20000 users should read 241-5701-030 *Passport 7400, 15000, 20000 Overview*.

You should have a good understanding of SNMP, and the objects in the Internet-standard Management Information Base (MIB II).

You should be familiar with provisioning and operational procedures, and have access to a standard Passport text interface.

How this document is organized

The 241-5701-300 *Passport 7400, 15000, 20000 SNMP Guide* contains the following sections:

What's new in this document

There were no new features added to this document.

Other changes made to this document include the following:

- The section “Generic error response to a set request” (page 241) was updated with information about displaying operational attributes under the *ProvisioningSystem* component.

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

  Passport commands are not case-sensitive and do not have to match commands and parameters exactly as shown in this document, with the exception of string options values (for example, file and directory names) and string attribute values.

- |

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

- ...

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

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

Related documents

See the following documents for related information:

- 241-5701-030 *Passport 7400, 15000, 20000 Overview*
- 241-5701-050 *Passport 7400, 15000, 20000 Commands*
- 241-5701-060 *Passport 7400, 15000, 20000 Components*
- 241-5701-500 *Passport 6400, 7400, 15000, 20000 Alarms*
- 241-5701-600 *Passport 7400, 15000, 20000 Configuration Guide*
- 241-5701-810 *Passport 7400, 15000, 20000 Configuring IP*

For information on last minute updates to the product, see the Release Notes for this PCR release.

How to get more help

For information on training, problem reporting, and technical support, see the “Nortel Networks support services” section in the product overview document.

Standards

Passport SNMP conforms to several standards. For a detailed list of the specific standards, see the table “Internet-Standard MIB compliance” (page 38).

- ISO 8824, *Information Processing Systems - Open Systems Interconnection Specification of Abstract Syntax Notation One (ASN.1)*, International Organization for Standardization. Dec 1987.
- RFC 1155, *Structure and Identification of Management Information for TCP/IP-based Internets*, M. Rose, Performance Systems International Inc., and K. McCloghrie, Hughes LAN Systems Inc., May 1990.
- RFC 1156, *Management Information Base for Network Management of TCP/IP-based internets*, K. McCloghrie, Hughes LAN Systems Inc., and M. Rose, Performance Systems International Inc., May 1990.
- RFC 1157, *A Simple Network Management Protocol (SNMP)*, J. Case, SNMP Research, M. Fedor and M. Schoffstall, Performance Systems International Inc., and J. Davin, MIT Laboratory for Computer Science, May 1990.
- RFC 1212, *Concise MIB Definitions*, M. Rose, Performance Systems International Inc., and K. McCloghrie, Hughes LAN Systems Inc., Editors, March 1991.
- RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB II*, K. McCloghrie, Hughes LAN Systems Inc., and M. Rose, Performance Systems International Inc., Editors, March 1991.
- RFC 1215, *A Convention for Defining Traps for use with the SNMP*, M. Rose, Performance Systems International Inc., Editor, March 1991.
- RFC 1902, *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.

- RFC 1903, *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.
- RFC 1904, *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.
- RFC 1905, *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.
- RFC 1906, *Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.
- RFC 1907, *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)*, SNMPv2 Working Group, J. Case, SNMP Research, Inc., K. McCloghrie, Cisco Systems, Inc., M. Rose, Dover Beach Consulting, Inc., S. Waldbusser, International Network Services, January 1996.
- RFC 2575, *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*, B. Wijnen, IBM,T. J. Watson Research, R. Presuhn, BMC Software, Inc., and K. McCloghrie, Cisco Systems, Inc., April 1999.

Chapter 1
SNMP configuration

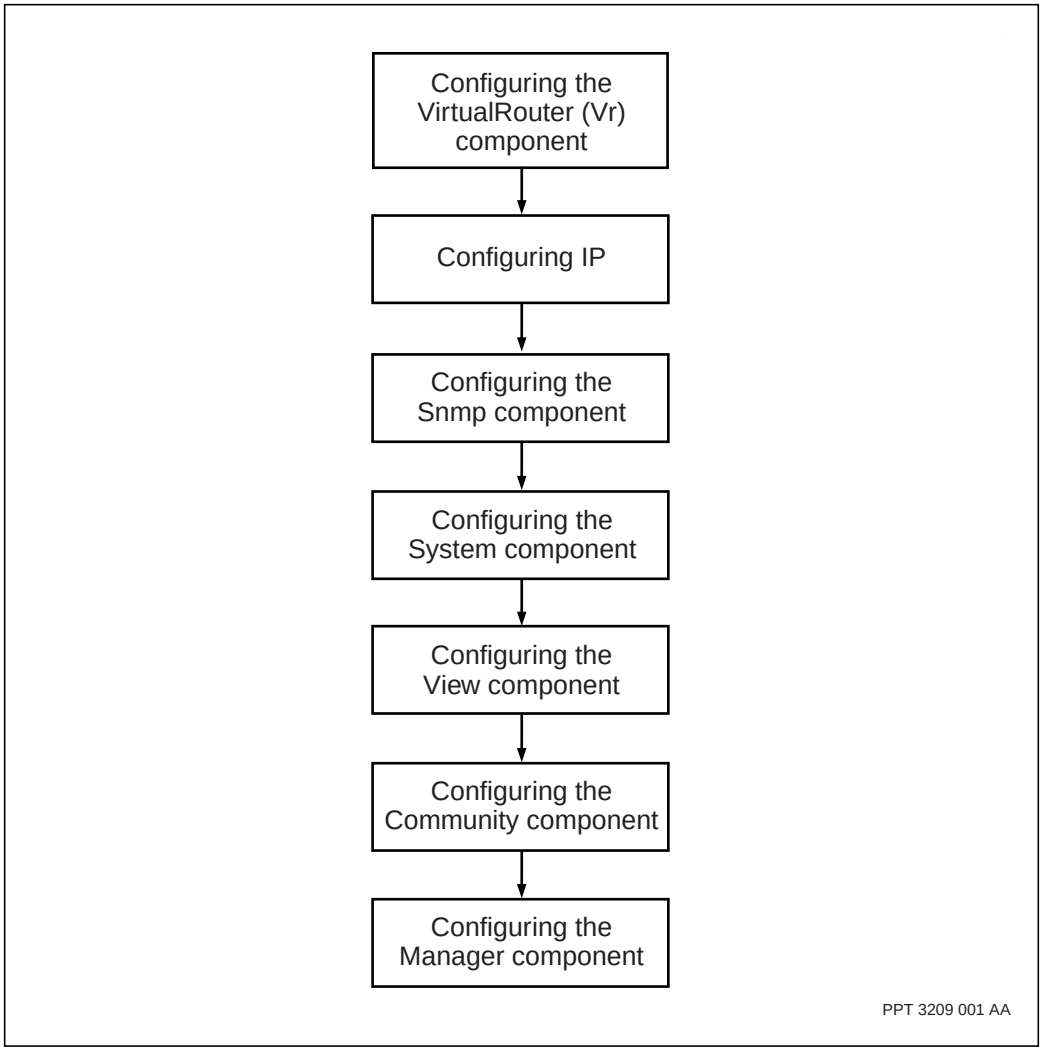
Passport SNMP is a set of network management protocols and functions that communicate using the Internet Protocol (IP) stack. You can use SNMP to perform the following tasks:

- integrate multi-vendor surveillance
- collect statistics for either real-time display or long-term storage
- perform simple control and configuration tasks, such as the disabling of a port, or the changing of a Passport trunk metric for influencing routing decisions

SNMP configuration task flow

This task flow shows you the sequence of procedures you perform to configure SNMP. To link to any procedure, go to “Navigation links” (page 24).

Figure 1
SNMP configuration task flow



Navigation links

- To configure the *VirtualRouter* (*Vr*) component, refer to 241-5701-271 *Passport 7400, 15000, 20000 Network Management Connectivity*.

- To configure IP on the *Vr* component, refer to 241-5701-271 *Passport 7400, 15000, 20000 Network Management Connectivity*.
- “Configuring the Snmp component” (page 26)
- “Configuring the System component” (page 28)
- “Configuring the View component” (page 30)
- “Configuring the Community component” (page 32)
- “Configuring the Manager component” (page 34)

Configuring the Snmp component

Configure the *Snmp* component (a subcomponent of the *VirtualRouter* (*Vr*) component) to set up an SNMP agent on a Passport. The SNMP agent must be configured before an SNMP manager can access a Passport switch. Configure the SNMP agent from either the local console, a telnet or an FMIP session.

Prerequisites

- Configure the *VirtualRouter* component as described in 241-5701-810 *Passport 7400, 15000, 20000 Configuring IP*.

  Note: Passport supports multiple virtual routers. The first virtual router you add is by default the management virtual router, and subsequent additions are customer virtual routers. If you only add one virtual router, you can configure it as a customer virtual router. For more information on multiple virtual routers, see "SNMP access on multiple virtual routers" (page 61).

- Configure the IP connectivity on the *Vr* component as described in 241-5701-810 *Passport 7400, 15000, 20000 Configuring IP*.

Procedure steps

1 Add the *Snmp* component under the *VirtualRouter* component.

```
add VirtualRouter/<n> Snmp
```

2 Specify the IP stack that the SNMP agent will monitor. This is determined by the IP connectivity.

```
set VirtualRouter/<n> Snmp ipStack <stack_value>
```

3 Specify whether traps are generated when requests are received with invalid community profiles.

```
set VirtualRouter/<n> Snmp enableAuthenTraps
<trap_value>
```

4 Specify whether Passport alarms are mapped to Passport enterprise traps.

```
set VirtualRouter/<n> Snmp alarmsAsTraps <alarm_value>
```

5 Specify whether the Passport customer identifier (CID) in the trap is enabled.

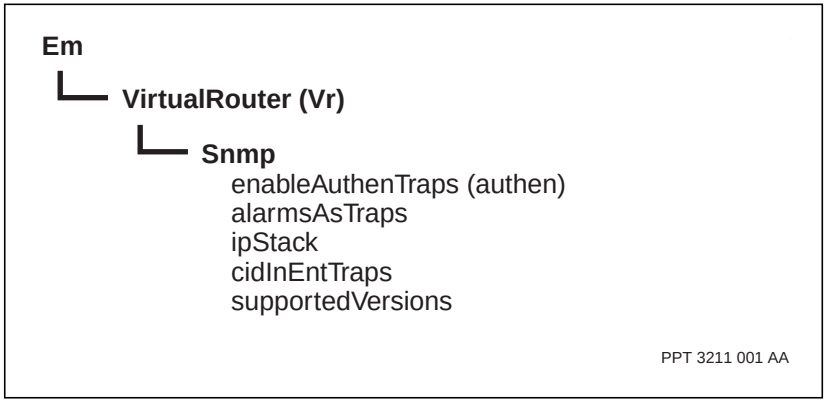
```
set VirtualRouter/<n> Snmp cidInEntTraps
<cid_trap_value>
```

Variable definitions

Variable	Value
<alarm_value>	The value for the *alarmsAsTraps* attribute. It can be set to enabled or disabled.
<cid_trap_value>	The value for the *cidInEntTraps* attribute. It can be set to enabled or disabled.
<n>	The instance of the *VirtualRouter* component.
<stack_value>	The value for the *ipStack* attribute. It can be set to either vrIp (Virtual Router IP stack) or ipiFrIpivc (IP interface over VC or over Frame Relay).
<trap_value>	The value for the *enableAuthenTraps* attribute. It can be set to enabled or disabled.

Procedure job aid

Figure 2
Configuring Snmp component hierarchy



Configuring the System component

Configure the *System (Sys)* component, which is a required subcomponent of the *Snmp* component. It is automatically added when you configure the *Snmp* component. Configuring the *Sys* component establishes the specific details for the SNMP agent you are configuring, including the system name, location, and contact name.

Procedure steps

1. Configure the system name for this particular Passport node.

   set VirtualRouter/<n> Snmp System name <system_name>

2. Configure the location of the system for this particular Passport node.

   set VirtualRouter/<n> Snmp System location <system_location>

3. Configure the contact name for this particular Passport node.

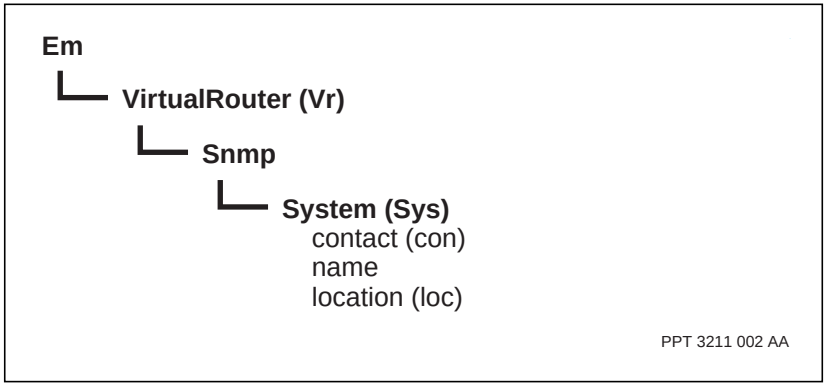
   set VirtualRouter/<n> Snmp System contact <system_contact>

Variable definitions

Variable	Value
<n>	The instance of the *VirtualRouter* component
<system_contact>	The value of the *contact* attribute. It identifies the contact person assigned to the particular Passport node, and how to contact them. It can be a string of ASCII text, and has no default value.
<system_location>	The value of the *location* attribute. It identifies the physical location of the particular Passport node. It can be a string of ASCII text, and has no default value.
<system_name>	The value of the *name* attribute. It identifies the administrative name assigned to the particular Passport virtual router. It can be a string of ASCII text, and has no default value.

Procedure job aid

Figure 3
Configuring System component hierarchy



Configuring the View component

Configure the *View* component as a required part of configuring community profiles. A community uses the view index to reference a collection of included and excluded MIB subtrees.

Procedure steps

1. Add one or more *View* components.

   ```
   add VirtualRouter/<n> Snmp View/<name,subtree>
   ```

2. Optionally, set the value of the *mask* attribute. This attribute, combined with the subTree value in the previous step, determines the family of view subtrees.

   ```
   set VirtualRouter/<n> Snmp View mask <mask_value>
   ```

3. Optionally, set the value of the *type* attribute. This determines whether the particular family of view subtrees are included or excluded in the community's MIB view.

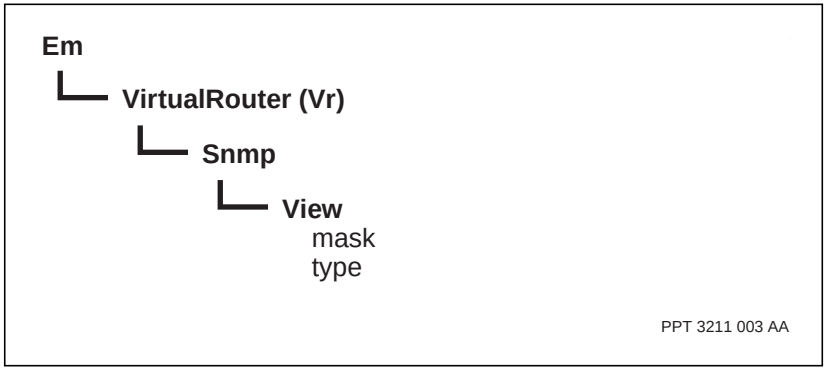
   ```
   set VirtualRouter/<n> Snmp View type <type_value>
   ```

Variable definitions

Variable	Value
<mask_value>	The value for the *mask* attribute. The default value is null.
<n>	The instance of the *VirtualRouter* component.
<name,subtree>	The indices for the *View* component: **name**: the index indicating the family of *View* components. **subTree**: the index indicating the MIB subtree to be included or excluded in this view.
<type_value>	The value for the *type* attribute. It can be set to includedSubtree (default value) or excludedSubtree.

Procedure job aid

Figure 4
Configuring View component hierarchy



Configuring the Community component

Configure the *Community* (*Com*) component as a required part of configuring community profiles. A community pairs up an SNMP agent to a set of SNMP management applications. The *Com* component represents either an SNMPv1 or SNMPv2 community.

Procedure steps

1. Add one or more *Community* components.

   `add VirtualRouter/<n> Snmp Community/<m>`

2. Set the *communityString* attribute, used to authenticate SNMP messages from SNMPv1 or SNMPv2c management entities.

   `set VirtualRouter/<n> Snmp Com/<m> communityString <community_name>`

3. Specify the set of MIB subtrees accessible by this community. This value must correspond to the value of the view index assigned to one of the View components.

   `set VirtualRouter/<n> Snmp Com/<m> viewIndex <name>`

4. Set the *accessMode* attribute, used to determine the level of access the community has for each object in its view.

   `set VirtualRouter/<n> Snmp Com/<m> accessMode <access_mode_value>`

Variable definitions

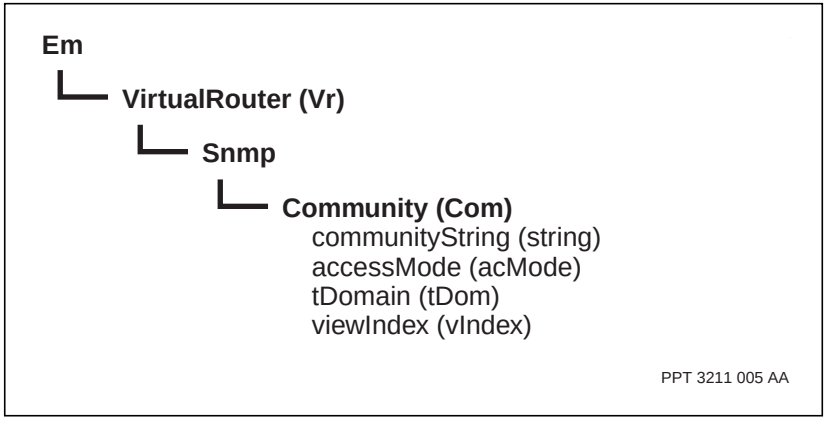
Variable	Value
<access_mode_value>	The value for the *accessMode* attribute. It can be set to readOnly (default value) or readWrite.
<community_name>	The value for the *communityString* attribute. The value of this attribute, combined with the value of the *tDomain* attribute (which is always set to snmpUdpDomain) must be unique across all *Community* components. The default value is public.
<m>	The instance of the *Community* component:

(Sheet 1 of 2)

Variable	Value
<n>	The instance of the *VirtualRouter* component.
<name>	The index indicating the family of *View* components. This value must correspond to the first index of one of the View components.
(Sheet 2 of 2)	

Procedure job aid

Figure 5
Configuring Community component hierarchy



Configuring the Manager component

Configure the *Manager* (*Man*) component to provide IP access checking, as well as a destination for generated traps. Adding a *Man* component is an optional step; if you do not add a *Man* component under a *Community* component, any SNMP manager may use that community profile.

Procedure steps

1. Add one or more *Manager* components.

   `add VirtualRouter/<n> Snmp Community/<m> Manager/<x>`

2. Set the *transportAddress* attribute, which is the SNMP manager's UDP transport address.

   `set VirtualRouter/<n> Snmp Com/<m> Man/<x> transportAddress <d.e.f.g-162>`

3. Set the privileges attribute, which specifies the access privileges governing what management options the SNMP manager can perform.

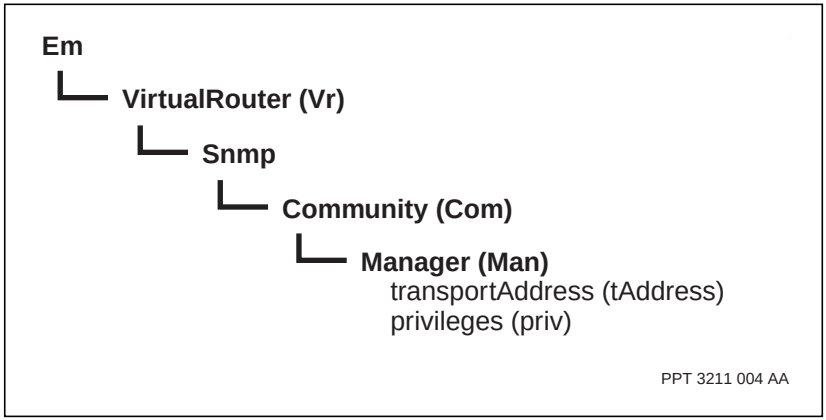
   `set VirtualRouter/<n> Snmp Com/<m> Man/<x> privileges <privilege_value>`

Variable definitions

Variable	Value
<d.e.f.g-162>	The value of the *transportAddress* attribute. The "a.b.c.d" indicates the IP address, and the "162" is the UDP port.
<m>	The instance of the *Community* component:
<n>	The instance of the *VirtualRouter* component.
<privilege_value>	The value for the *privilege* attribute. It can be set to one of gets (default value), sets, v1trap, or v2trap. The sets access privilege is only available if the *accessMode* attribute is set to readWrite. ***Note:*** You can specify only one of either SNMPv1 or SNMPv2 trap formats to be sent to the management application.
<x>	The instance of the *Manager* component:

Procedure job aid

Figure 6
Configuring Manager component hierarchy



Chapter 2
Passport SNMP fundamentals

Under SNMP, managed entities (such as routers, concentrators, and host computers) contain software components called agents which monitor the operation of the managed entity. To do this, the agent maintains a collection of objects in the management information base (MIB). The MIB reflects the operation of the managed entity.

A manager application, which normally runs on a network station, exchanges messages with the agent to access the agent's MIB. The manager reads from, and writes to, objects in the MIB according to predefined access privileges that have been assigned to the MIB objects. SNMP defines the protocols and message formats that are used to perform the read and write operations.

When the manager sets a MIB object to a new value, the agent may respond by changing the operation of the managed entity in some way. The agent does not normally transmit the contents of its MIB objects without first being queried by the manager. However, should an exceptional event occur, an alarm, called a trap, is raised. A trap is an unsolicited message containing information about the event.

A manager must know the structure of the MIB that the agent implements to understand object names, object types, what the object represents, and the permissible types of access to the objects. This structural information is provided to the manager by using Internet-standard MIBs and enterprise MIBs.

Passport SNMP Version 2c supports both SNMPv1 and SNMPv2c and conforms to appropriate standards as indicated throughout this guide. For details on Passport SNMP performance and engineering guidelines, contact your Nortel Networks representative.

In looking at SNMP fundamentals, refer to the following:

- "Standard MIBs" (page 38)
- "Enterprise MIBs" (page 40)
- "SNMP Administrative model" (page 41)

Standard MIBs

Standard MIBs are MIBs defined by the Internet community. The table, "Internet-Standard MIB compliance" (page 38) provides a summary of the Standard MIBs supported by the Passport system. The precise level of support for each Standard MIB is documented through agent capabilities statements (see "Passport Enterprise MIBs" (page 71)).

Please see "Retrieving Standard MIBs" (page 219) for details on how to retrieve Standard MIBs for use by a network manager.

Table 1
Internet-Standard MIB compliance

RFC	Name
RFC 1155	Structure and Identification of Management Information for TCP/IP-based internets
RFC 1157	A Simple Network Management Protocol (SNMP)
RFC 1212	Concise MIB Definitions
RFC 1213	Management Information Base for Network Management of TCP/IP-based internets: MIB-II
RFC 1253	OSPF Version 2 Management Information Base (Note 1 and Note 3)
RFC 1315	Management Information Base for Frame Relay DTEs (Note 2)

(Sheet 1 of 3)

Table 1 (continued)
Internet-Standard MIB compliance

RFC	Name
RFC 1354	IP Forwarding Table MIB
RFC 1471	The Definitions of Managed Objects for the Link Control Protocol of the Point-to-Point Protocol
RFC 1472	The Definitions of Managed Objects for the Security Protocols of the Point-to-Point Protocol
RFC 1473	The Definitions of Managed Objects for the IP Network Protocol of the Point-to-Point Protocol
RFC 1573	Evolution of the Interfaces Group of MIB II (see Note 3)
RFC 1643	Definitions of Managed Objects for the Ethernet-like Interface Types
RFC 1657	Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2
RFC 1748	IEEE 802.5 MIB using SMIv2 (see Note 3)
RFC 1749	IEEE 802.5 Station Source Routing MIB using SMIv2 (see Note 3)
RFC 1902	Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)
RFC 1903	Textual Conventions for Version 2 of the Simple Network Management Protocol
RFC 1904	Conformance Statements for Version 2 of the Simple Network Management Protocol
RFC 1905	Protocol Operations for Version 2 of the Simple Network Management Protocol
RFC 1906	Transport Mappings for Version 2 of the Simple Network Management Protocol

(Sheet 2 of 3)

Table 1 (continued)
Internet-Standard MIB compliance

RFC	Name
RFC 1907	Management Information Base for Version 2 of the Simple Network management Protocol
Note 1: Actual implementation is a draft of the evolution of RFC 1253: draft-ietf-ospf-mib-04.txt	
Note 2: Actual implementation is a draft of the evolution of RFC 1315: draft-ietf-iplpdn-frmib-dte-02.txt.	
Note 3: These RFC standards are written in SMIv2 format. If your manager is SMIv1 only, you must convert the required RFCs to SMIv1 format. See "Retrieving Standard MIBs" (page 219).	

(Sheet 3 of 3)

All mandatory variables in a supported Standard MIB can not always be implemented. Unsupported mandatory variables are handled as recommended by the relevant Internet authorities.

Enterprise MIBs

Enterprise MIBs are used for surveillance of the Passport switch. The Standard MIBs supplement the use of the Enterprise MIBs.

The Passport Enterprise MIBs are used to meet the requirements of a variety of classes of SNMP-based network management systems (managers), each exhibiting different patterns of use. An example is a simple manager who needs to poll certain state variables and counters, and who also needs to set certain Standard and Enterprise MIB variables.

The Passport Enterprise MIBs have the following characteristics:

- modular content – The Passport Enterprise MIBs are partitioned into MIB modules, each specifying the management information associated with some aspect of a Passport feature. For example, the *FileSystemMIB* provides information concerning the File System, the *FrameRelayUniMIB* provides information concerning the Frame Relay UNI feature.

- capability – Passport Enterprise MIB capabilities are also provided in a modular fashion. For example, with respect to fault management, the Passport Enterprise MIBs alarm reporting facility need not be used. Traditional SNMP polling mechanisms are always available; the value-added Passport Enterprise MIBs fault management capabilities do not complicate or interfere with the use of simpler, well-known mechanisms.
- consistency – All MIB modules and all MIB capabilities are provided in a consistent manner across the entire MIB.
- standards-based functionality – Standard mechanisms are typically used to provide required functionality. For example, the *RowStatus* textual convention supports table row addition and deletion. The *ORTable* supports MIB discovery.

Passport Enterprise MIBs reflect the Passport proprietary component data model. Refer to "Passport Enterprise MIBs" (page 71) for more information on Passport Enterprise MIBs and the Passport component model.

SNMP Administrative model

Operations of the SNMP protocol are carried out under an administrative framework consisting of communities and access modes. In Passport, this framework is augmented by the implementation of a view mechanism which provides access control to both Standard and Enterprise MIBs. For information on configuring SNMP on Passport, refer to "SNMP configuration" (page 23).

Model constraints

Certain constraints are applied to the Administrative model. The table "Constraints on the administrative model" (page 42), shows these constraints.

Table 2
Constraints on the administrative model

Administrative model	Constraint
maximum received PDU size	1 472 octets
maximum transmitted PDU size	1 472 octets
maximum number of views	1 024
maximum number of communities	1 024
maximum number of managers per community	100
maximum length of a community string	255
maximum length of a view name	32

Communities

A pairing of a SNMP agent with a set of SNMP managers is called a SNMP community. Each community is named by a string of octets that is called its community name. The term community string is used to refer to the ASCII encoded representation of the name. These terms are often used interchangeably.

The community string is placed in SNMP messages sent out by managers and agents and is used by recipients to authenticate these messages. When a SNMP message is received, the community string is checked to ensure that it matches one locally configured. The string public is commonly used.

Views

For security reasons it is often valuable to restrict the access rights of some management applications to a specified subset of the information available from a node.

Access to variables is controlled by MIB views which identify specific sets of MIB variables. For example, for a given node, there is typically one MIB view which provides access to all management information, and often there are other views, each of which contains some subset of the information. The management application is restricted by providing access rights in terms of the MIB views it can access.

Since variables are identified through the MIB structure, it is convenient to define a MIB view as a combination of a set of view subtrees within the MIB structure. A simple MIB view, for example, all *mscFrUni* variables, can be defined as a single subtree while more complicated MIB views, for example, all information relevant to a particular network interface, can be represented by the union of multiple subtrees.

Any set of variables can be described by the union of some number of subtrees. For example, when specifying all columns in one particular row of a MIB table, they appear in separate subtrees, one per column, each with a very similar format. Because the formats are similar, the required set of subtrees can easily be aggregated into one structure. This structure is called a masked subtree. A masked subtree can either be included or excluded from a MIB view. The view table is used to define the set of subtrees which are included in and those are excluded from each MIB view. Refer to “Masked view subtrees” (page 45) for more details.

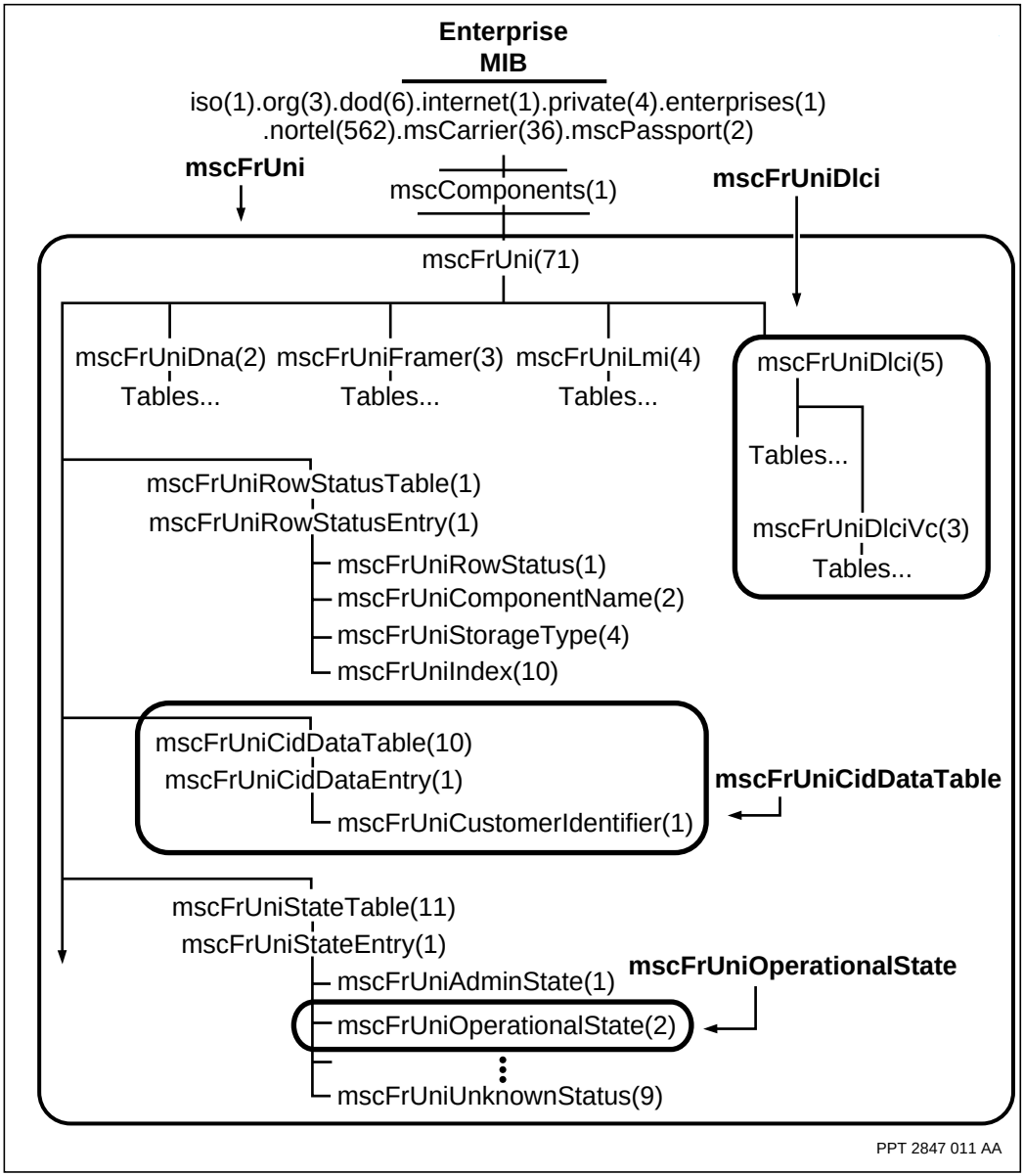
View subtrees

A view subtree is the set of all MIB variables contained within a subtree of the MIB structure. These variables all have a common OBJECT IDENTIFIER prefix to their names, which is the OBJECT IDENTIFIER value of the root of the subtree. This value is called the view subtree name.

The figure, “View subtrees in the mscFrUni subtree” (page 44), illustrates the following view subtrees and their names:

- *mscFrUni*: scopes all variables of all tables in the *mscFrUni* component and its subcomponents
- *mscFrUniDlci*: scopes all variables of all tables in the *mscFrUniDlci* component and its subcomponents
- *mscFrUniCidDataTable*: scopes all variables in the *mscFrUniCidDataTable*
- *mscFrUniOperationalState*: scopes the *mscFrUniOperationalState* variable

Figure 7
View subtrees in the mscFrUni subtree



The figure, “View subtrees in the mscFrUni subtree” (page 44), illustrates the mapping of the hierarchial structure of the Passport component information tree on the MIB structure. This mapping eases the specification of view subtrees.

New variables added to tables, new tables added to components and new subcomponents added to component hierarchies are automatically included in existing subtree specifications. This eases the maintenance tasks associated with MIB evolution. That is, the specification of view subtrees typically do not have to be modified as the MIB evolves.

> ***Note:*** If the OBJECT IDENTIFIER prefix identifying a view subtree is longer than the OBJECT IDENTIFIER of a variable type, the view subtree for access control has granularity at the instance level. This level of granularity is not supported.

Masked view subtrees

A bit string value, or mask, is applied to a view subtree to further refine the identified set of variables. The mask indicates which arcs of the associated view subtree are significant.

A variable is scoped by a masked subtree if both of the following statements are true:

- The OBJECT IDENTIFIER name of the variable contains at least as many arcs as does the view subtree name.
- Each sub-identifier in the OBJECT IDENTIFIER name of the variable matches the corresponding arc of the view subtree name whenever the corresponding bit of the mask is non-zero.

When the configured value of the mask is all ones the masked subtree is identical to the view subtree.

When the configured value of the mask is shorter than required to perform the above test its value is implicitly extended with ones. Consequently, a masked view subtree having a mask of zero length is also identical to the view subtree. For example, the mask ‘’H is identical to the mask ‘FF’H.

Refer to the tables “Masked subtree (Example 1)” (page 46) and “Masked subtree (Example 2)” (page 46).

Table 3
Masked subtree (Example 1)

Subtree name	Mask	
mscLp	‘’H	Label
1.3.6.1.4.1.562.36.2.1.12	‘’H	Value

Note: The ‘’H mask refers to a hexadecimal mask. By default, a hexadecimal mask with only ‘’H implies a null mask (all ones).

The name and mask look like this:

```
Mask: 1.1.1.1.1.1.1.1.1.1...
Name: 1.3.6.1.4.1.562.36.2.1.12
```

This masked subtree is the same as the subtree *mscLp*. This subtree scopes all of the tables for *Lp* and all of its subcomponents.

Table 4
Masked subtree (Example 2)

Subtree name	Mask	
mscComponents.0.1	‘FFDF’H	Label
1.3.6.1.4.1.562.36.2.1.0.1	‘FFDF’H	Value

This example looks like this:

```
Mask: 1.1.1.1.1.1.1.1.1.0.111...
Name: 1.3.6.1.4.1.562.36.2.1.0.1
```

where: 1.1.1.1 in the binary mask is equal to F in hexadecimal and 1.1.0.1 in the binary mask is equal to D in hexadecimal.

This masked subtree exploits the *RowStatusTable* that is always assigned the arc *1* to define a masked subtree which scopes only the *RowStatusTable* for first-level components.

MIB view specification

A MIB view is specified as a collection of masked view subtrees with each masked view subtree defined to be either included in, or excluded from, the MIB view. View subtrees defined with the same view index belong to the same MIB view.

For example, a MIB view can be defined which contains everything in the *mscLp* subtree except for the *mscLpDs3* tables.

A variable is contained within the MIB view according to its masked subtrees.

- If the variable belongs to none of the masked subtrees the variable is not in the MIB view. For example, *mscFrUni* is not in the MIB view shown in the table "MIB view specification example" (page 47).
- If the variable belongs to exactly one of the masked subtrees, then the variable is included in, or excluded from, the MIB view according to the type of that subtree. In the table "MIB view specification example" (page 47), the *mscLpRowStatusTable* belongs only to the *mscLp* subtree and therefore included.
- If a variable belongs to more than one of the relevant masked subtrees, then the variable is either included or excluded from the MIB view according to the type of subtree which best scopes it. The scoping subtree is the one with which it places the greatest number of sub-identifiers, and whose subtree name is lexicographically greater. In the table "MIB view specification example" (page 47), the *mscLpDS3ProvTable* belongs to both the *mscLp* and *mscLpDS3* masked subtrees. As the *mscLpDS3* masked subtree shares the greatest number of sub-identifiers with the *mscLpDS3ProvTable*, the *mscLpDS3ProvTable* is scoped by *mscLpDS3* subtree and is excluded from the MIB view.

Table 5
MIB view specification example

View	Subtree name	Mask	Type
3	mscLp	'FF'H	included
3	mscLpDS3	'FF'H	excluded

Controlling event reporting

A Passport agent can be configured to send no traps, only standard traps, only Enterprise traps or both Standard and Enterprise traps to its authorized managers.

A Passport agent converts an alarm into a trap that is sent to managers if the SNMP component has the *alarmsAsTraps* attribute enabled. A Passport agent sends a standard trap or Enterprise trap (for example, an alarm) to a manager when a corresponding event occurs if the agent is configured so that:

- The manager belongs to a community configured on the node.
- The manager's transport address is properly configured and the manager is granted *v1trap* or *v2trap* privileges for that community.

The operational model for event reporting is illustrated in the figure "Event reporting" (page 49). There are three steps:

1 Managers receiving traps must be configured to display, interpret and/or log them. The procedures involved vary for different types of managers

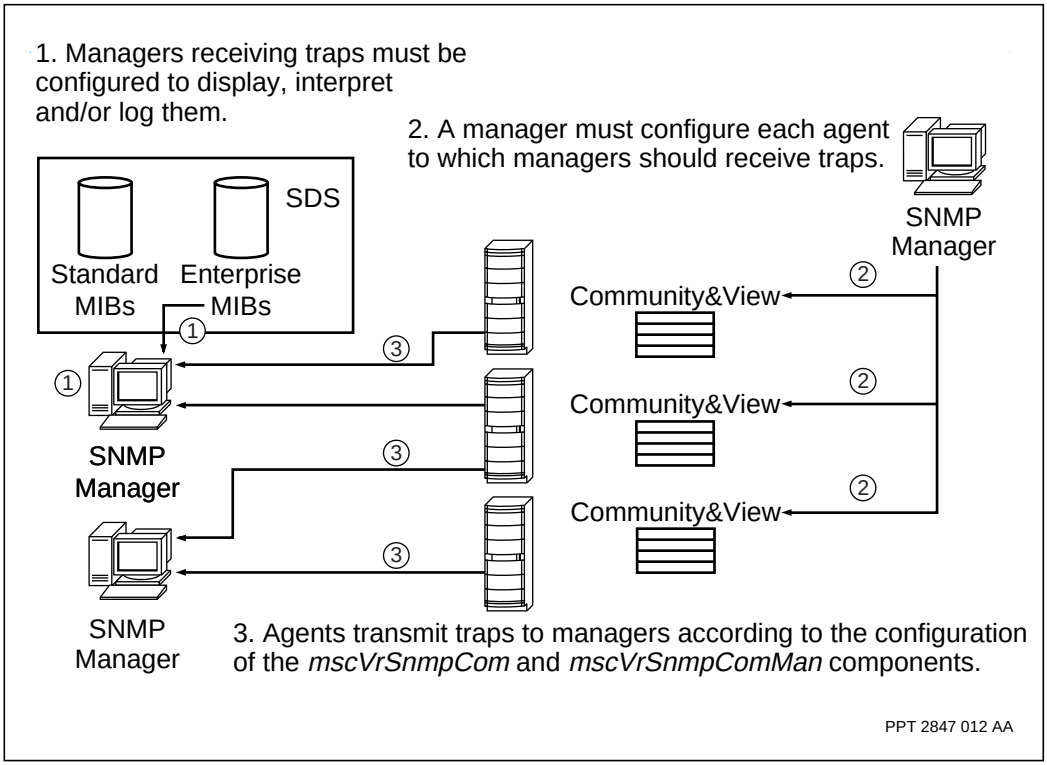
2 A manager must provision the *Vr Snmp Com*, *Vr Snmp View* and *Vr Snmp Com Man* components on each node.

Note: More details are provided further in this chapter.

3 The agents then send traps to targeted managers, according to the configuration of these components.

Note: An agent does not send traps unless it is configured to do so.

Figure 8
Event reporting



Enterprise trap

To facilitate configuration of management statistics, to display, interpret, and log Enterprise-defined traps, all Enterprise traps are defined under the *mscPassportTraps* object. Refer to the *nortelPC-alarmV1_Cxxx.mib* for more information on Enterprise Traps.

SNMP component hierarchy

Some important aspects of the SNMP MIB tree are illustrated in the figure "Passport SNMP MIB tree overview" (page 50). The following components are of concern for SNMPv1 or SNMPv2c provisioning:

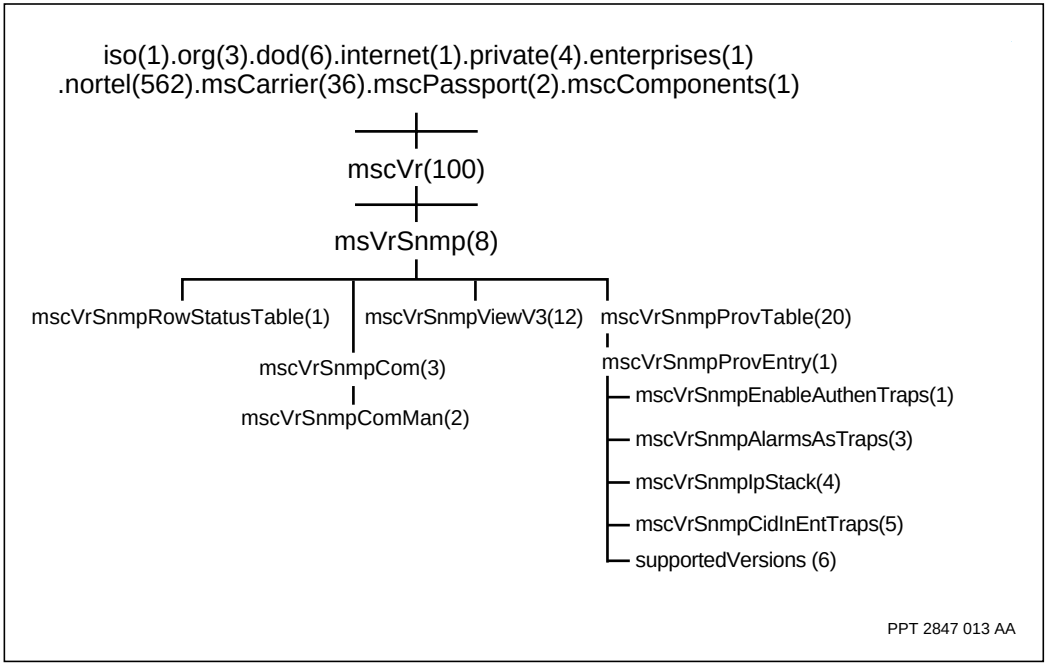
- the SNMP component (*mscVrSnmp*) which forms the root of the hierarchy

- the community component (*mscVrSnmpCom*) providing information concerning the communities known to the node
- a manager component (*mscVrSnmpComMan*) providing information concerning the managers associated with particular communities
- a View component (*mscVrSnmpView*) providing information concerning the views known to the node

This hierarchy contains all management information used to realize the administrative model described in this chapter.

Note: The SNMP component hierarchy is part of the *virtual router (mscVr*) component hierarchy.

Figure 9
Passport SNMP MIB tree overview



SNMP component (mscVrSnmp)

The *mscVrSnmp* subtree contains all provisionable and operational data relating to the SNMP management of the node. The *mscVrSnmp* subtree contains, in the *mscVrSnmpProvTable*, the global provisioning information associated with the agent. Five provisionable variables are defined as follows:

- *mscVrSnmpAuthenTraps* indicates whether the agent must generate SNMPv1 authentication traps
- *mscVrSnmpAlarmsAsTraps* indicates whether Passport 7400, 15000 alarms are sent as SNMP traps to SNMP managers
- *mscVrSnmpCidInEntTraps* controls whether the Customer Identifier (CID) is included in the Enterprise traps or not
- *mscVrSnmpIpStack* controls whether SNMP traffic uses the *Udp/Ip* provided by the *Vr Ip* feature or by the *Ipifr* feature
- *supportedVersions* indicates which SNMP message versions are accepted and sent by the SNMP agent

The tables of the *mscVrSnmp* subtree are indexed by *mscVrIndex*, of type *AsciiStringIndex*, and *mscVrSnmpIndex*, of type *NonReplicated*.

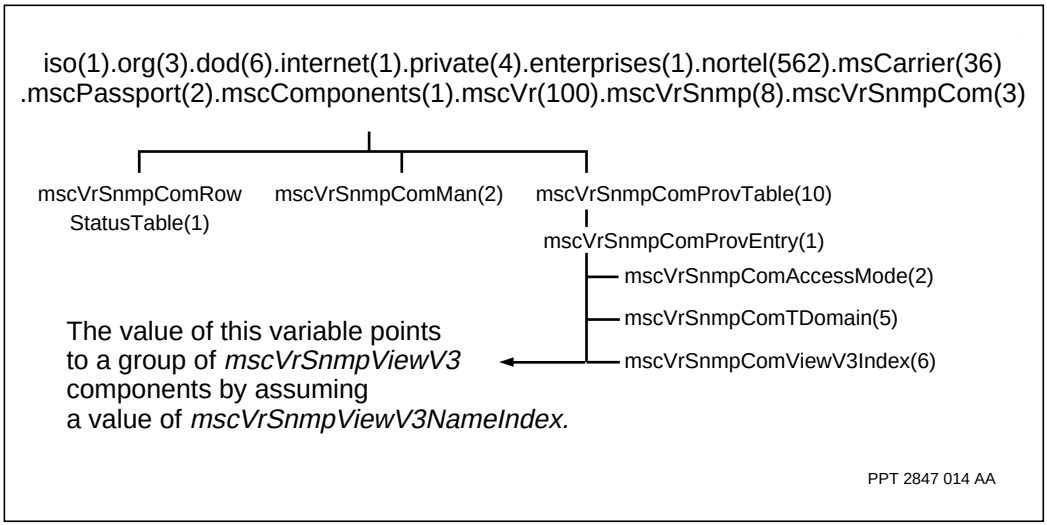
Community component (mscVrSnmpCom)

The *mscVrSnmpCom* subtree contains all provisionable and operational data concerning the SNMP communities known to the node. The *mscVrSnmpCom* subtree contains, in the *mscVrSnmpComProvTable*, the provisioning information associated with each community. The following provisionable variables are defined:

- *mscVrSnmpComAccessMode*: the maximum access mode for the community; one of readOnly or readWrite
- *mscVrSnmpComTDomain*: indicates the kind of transport service by which the community receives and sends network management traffic. The combination of *communityString* and *tDomain* attributes must be unique across all *VrSnmp* communities.
- *mscVrSnmpComViewV3Index*: the MIB view of this community. This value is the same as the value of *mscVrSnmpViewV3NameIndex*. That is, this value points to a family of configured MIB views using the value of *mscVrSnmpViewV3NameIndex*.

The MIB structure for the community component is shown in the figure "Community component" (page 52).

Figure 10
Community component



The *mscVrSnmpComCommunityString* variable must be unique across all communities.

The value of the *mscVrSnmpComCommunityString* variable is used to authenticate SNMP messages received from SNMPv1 or SNMPv2 managers and is placed in SNMP traps sent out by the agent.

When a message is received the community string in the message is checked to ensure that it matches a community string provisioned in *mscVrSnmpComCommunityString*. If there are no manager subcomponents for that community then the level of access is determined by the *mscVrSnmpComAccessMode* variable.

This is applicable for both SNMPv1 and SNMPv2 messages.

If there are manager subcomponents for the community, then the originating transport address in the message must match the transport address of one of these subcomponents and the level of access of the manager is determined by the *privileges* attribute for that manager.

The *mscVrSnmpCom subtree* has one index in addition to *mscVrIndex* and *mscVrSnmpIndex*. This is the *mscVrSnmpComIndex* which has a value of type *Integer32* identifying the community.

Manager component (mscVrSnmpComMan)

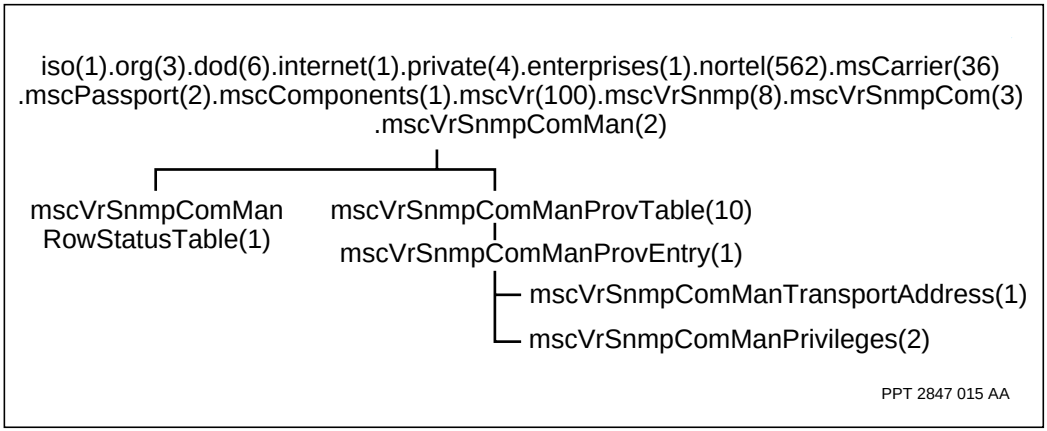
The *mscVrSnmpComManProvTable* of the *mscVrSnmpComMan* subtree contains provisioning information associated with a manager. The following provisionable variables are defined:

- *mscVrSnmpComManTransportAddress*: a transport address to which the agent sends SNMPv1 or SNMPv2 traps
- *mscVrSnmpComManPrivileges:* the access privileges granted by the SNMP manager, including *sets*, *gets*, *v1trap* or *v2trap* which are independently granted. The *sets* privilege can be granted only if the *mscVrSnmpComAccessMode* variable of the superior community component is set to readWrite. Any given manager can be provisioned with either *v1trap* or *v2trap*, not both.

  Note: A manager can only receive v1traps or v2traps if the v1trap or v2trap privileges are provisioned on the switch.

The MIB structure for the *Manager* component is shown in the figure “Passport Manager component” (page 54).

Figure 11
Passport Manager component



This component defines the access privileges for a SNMPv1 or SNMPv2 manager within the community defined by the superior community component. If a community has no subordinate manager subcomponents

- SNMPv1 or SNMPv2 messages are accepted from all source addresses using the proper community string dependent upon the provisioning of the SNMP *supportedVersions*.
- No *trap*s are sent out using this community string

If at least one manager subcomponent exists for a community, SNMPv1 or SNMPv2 messages containing that community string are only accepted if they originate from a transport address that matches the *mscVrSnmpComManTransportAddress* of one of these subcomponents.

The value of *mscVrSnmpComManTransportAddress* must be unique for all manager instances under the same community.

The *mscVrSnmpComMan* subtree has the *mscVrSnmpComManIndex* index in addition to *mscVrIndex, mscVrSnmpIndex* and *mscVrSnmpComIndex*. The *mscVrSnmpComManIndex* index has an *Integer32* value identifying the manager relative to the community.

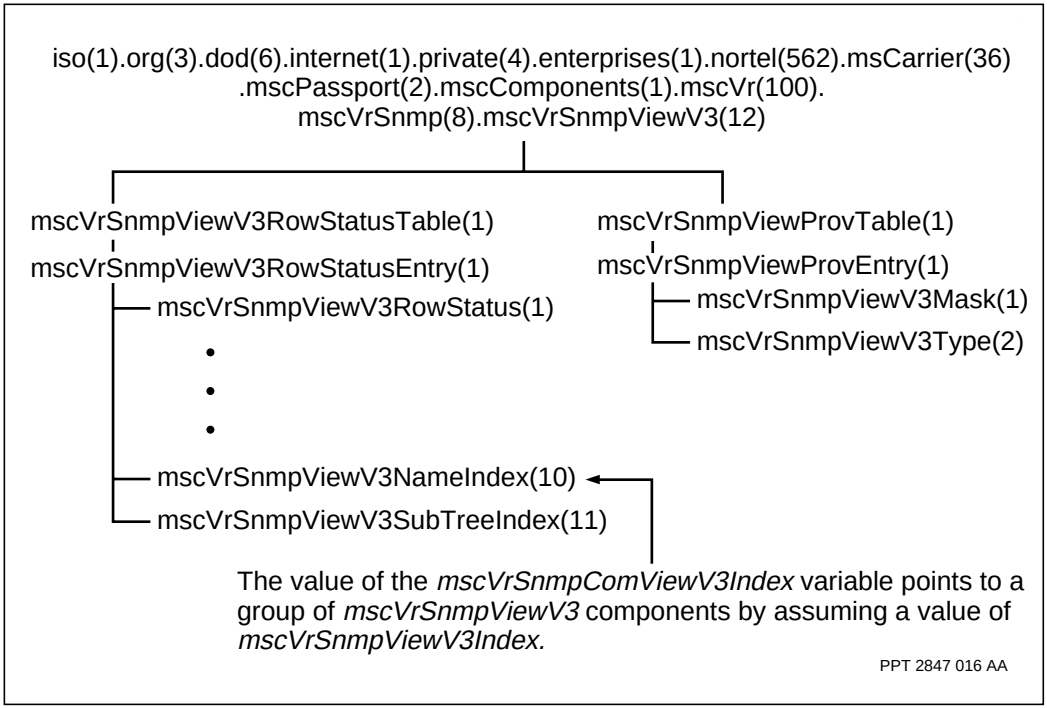
View component (mscVrSnmpViewV3)

The *mscVrSnmpViewV3* subtree contains, in the *mscVrSnmpViewV3ProvTable*, the provisioning information associated with a view. The following provisionable variables are defined:

- *mscVrSnmpViewV3Mask*: the view mask
- *mscVrSnmpViewV3Type:* the view type is *included* or *excluded*

The MIB structure for the *View* component is shown in the figure "Passport View component" (page 55).

Figure 12
Passport View component



The *mscVrSnmpViewV3* subtree has two indices in addition to *mscVrIndex* and *mscVrSnmpIndex*:

- *mscVrSnmpViewV3NameIndex:* a STRING value identifying the MIB view to which the subtree belongs

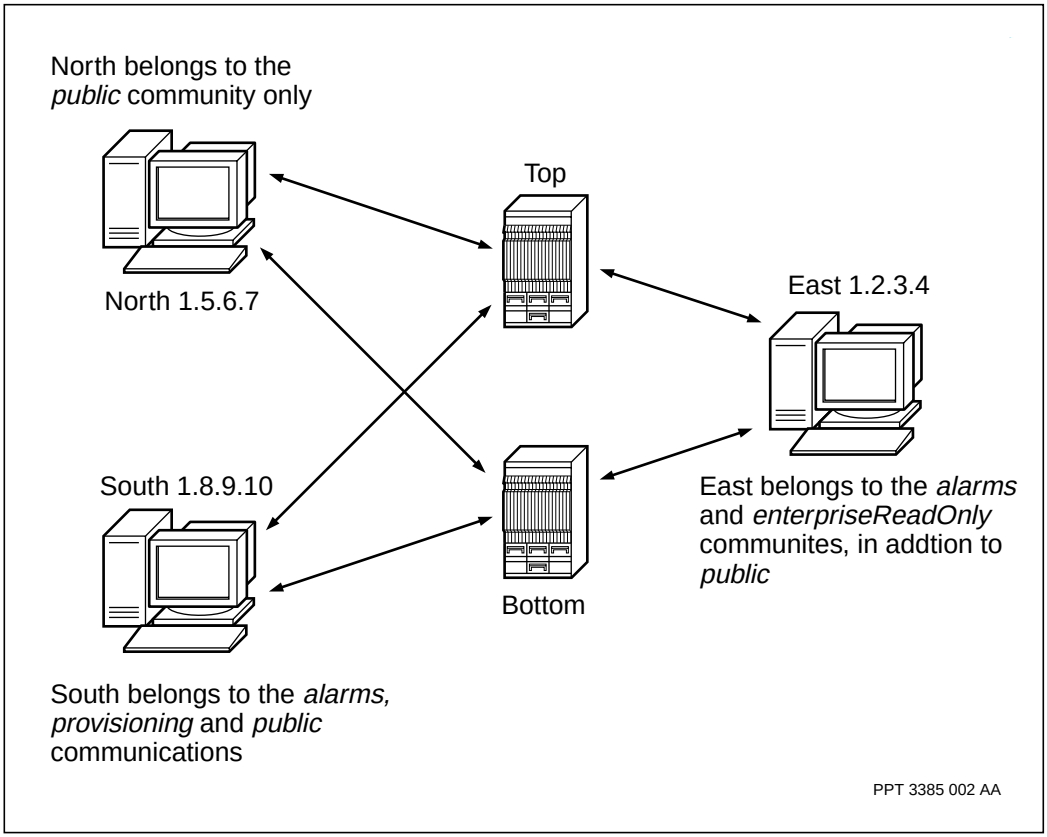
- *mscVrSnmpViewV3SubtreeIndex:* an OBJECT IDENTIFIER value naming the subtree, as described in “View subtrees” (page 43)

Example configuration

A simple management environment is illustrated in the figure “Example configuration” (page 57). There are four communities defined on both nodes:

- public, providing readOnly access to Standard MIBs only
- enterpriseReadOnly, providing readOnly access to both Standard and Enterprise MIBs
- provisioning: providing readWrite access to the Enterprise MIB
- alarms: used to send Passport Enterprise alarms

Figure 13
Example configuration



Example communities

The contents of the *mscVrSnmpComProvTable* on node Top and node Bottom are shown in the table "Example community view of node Top and node Bottom" (page 58).

Table 6
Example community view of node Top and node Bottom

Community	CommunityString	AccessMode	ViewIndex
1	public	readOnly	1
2	enterpriseReadOnly	readOnly	2
3	provisioning	readWrite	3
4	alarms	readWrite	3

Note: Since the value of *mscVrSnmpComTDom* is always set to *snmpUdpDomain* it is not shown in the above table.

Example managers

The content of the *mscVrSnmpComManProvTable* on both node Top and node Bottom is shown in the table "Example view of node Top and node Bottom" (page 59).

Table 7
Example manager view of node Top and node Bottom

Community	Manager	TAddr	Priv
2 (enterpriseReadOnly)	2	1.2.3.4-162 (east)	gets, ~sets, ~v1trap, ~v2trap
3 (provisioning)	1	1.8.9.10-162 (south)	gets, sets, v1trap
3 (provisioning)	1	1.8.9.10-162 (south)	gets, sets, v2trap
4 (alarms)	1	1.8.9.10-162 (south)	v1trap, ~gets, ~sets
4 (alarms)	2	1.2.3.4-162 (east)	v2trap, ~gets, ~sets

Note: There is no manager associated with the *public* community (community 1), which means all managers have access to this community's view.

Example views

The content of the *mscVrSnmpViewV3ProvTable* on both Top and Bottom is displayed in the table “Example view of node Top and node Bottom” (page 59). The *mgmt* subtree name is the OBJECT IDENTIFIER subtree scoping all Standard MIBs (*1.3.6.1.2*).

Table 8
Example view of node Top and node Bottom

View	Subtree name	Mask	Type	OBJECT IDENTIFIER
1	mgmt	‘FF’H	included	1.3.6.1.2
2	mscComponents	‘FF’H	included	1.3.6.1.4.1.562.36.2.1.3.2
3	mscComponents	‘FF’H	included	1.3.6.1.4.1.562.36.2.1.3.2
3	mgmt	‘FF’H	included	1.3.6.1.2

Chapter 3
SNMP access on multiple virtual routers

This section describes simple network management protocol (SNMP) access on multiple virtual routers (MVR), including information on:

The MVR feature implements multiple instances of router software on a single Passport switch. SNMP access on MVR allows you to provision an SNMP agent for each virtual router (VR). You can provision separate yet dedicated VRs on a single Passport switch. The SNMP access provides a management interface for each router allowing each router to be managed individually. For information on configuring virtual routers, refer to 241-5701-810 *Passport 7400, 15000, 20000 Configuring IP*.

Types of virtual routers

There are two types of virtual routers on a Passport switch.

- management (one only)
- customer

You can only provision one management VR on each Passport switch. The first VR you add is by default the management VR (but you can configure it as a customer VR if it is the only VR you are adding). You can also choose to provision a number of customer VRs.

Note: The maximum number of SNMP Agents supported on the Passport is eleven; therefore, only eleven SNMP agents can be provisioned and activated on the Passport.

The networking services available for the management and customer VRs differ somewhat. See 241-5701-805 *Passport 7400, 15000, 20000 Understanding IP* for a detailed description of these differences and a general description of the MVR feature.

Benefits of SNMP access on multiple virtual routers

SNMP access on MVR provides the following benefits:

- All VRs are SNMP manageable.
- VRs are individually manageable.
- SNMP management of the VRs includes
  - surveillance using SNMP *get*, *getNext*, and *getBulk* requests
  - configuration using SNMP *set* requests
  - discovery of the VRs as devices within an SNMP manageable map
- Restriction of the customer VR SNMP access.

Note: The SNMP manageable objects on a customer VR include only those components which are subcomponents of that VR or components associated with that VR through a link to the protocol port(s) of that customer VR.

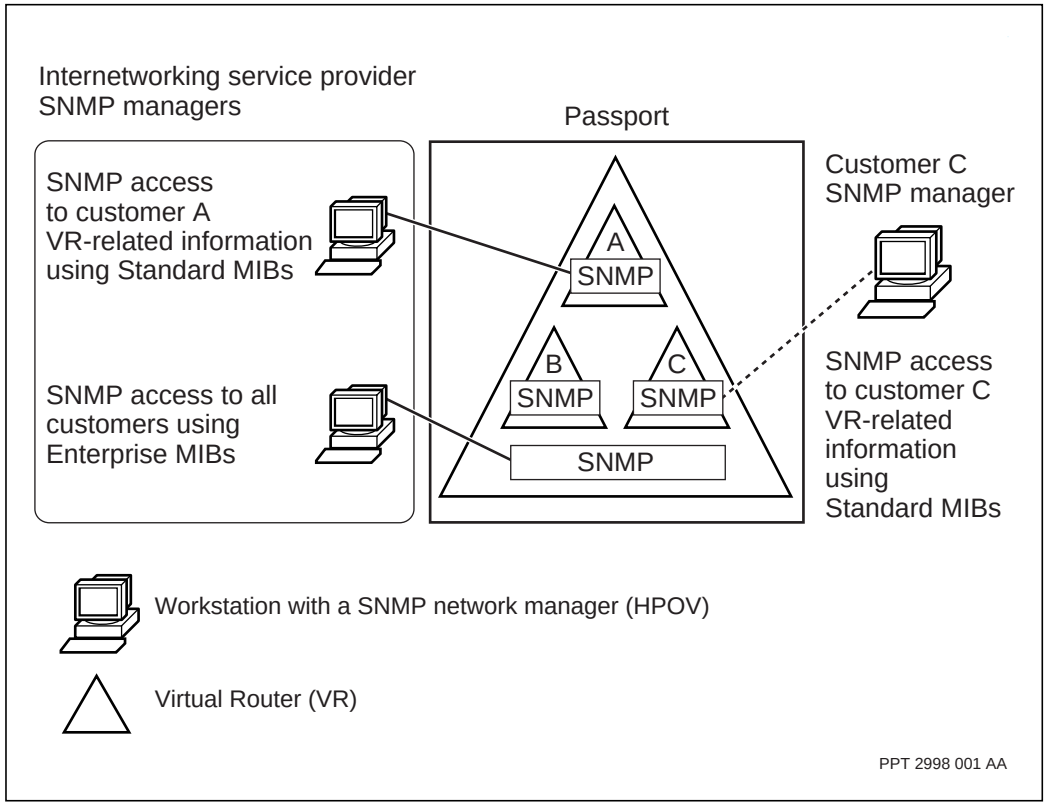
SNMP access capabilities

The SNMP access capabilities for the management and customer VRs differ considerably. The management VR contains SNMP access to all the services and applications on the Passport switch. The SNMP access available on the customer VRs only allow management of that VR, not other non-VR related

Passport services (for example, frame relay). The figure, “SNMP access from management and customer virtual routers” (page 63), illustrates the SNMP access for MVR architecture.

The internetworking service provider can optionally provision the SNMP access on the customer VRs to either allow or disallow SNMP access to the customer VR by an end-customer SNMP manager.

Figure 14
SNMP access from management and customer virtual routers



The SNMP access capabilities for the management VR include

- access to Enterprise MIBs
- access to the Standard MIB information for the management VR
- access to the Standard MIBs for non-VR related Passport network services
- access to ifTable entries for non-VR related Passport network services
- access to ifTable entries for the management VR
- receipt of all Enterprise traps
- receipt of Standard traps relating to the management VR
- receipt of Standard traps associated with non-VR related Passport network services

The SNMP access capabilities for the customer VRs include

- access to the Standard MIBs for the customer VR
- access to ifTable entries for the customer VR
- receipt of Standard traps for the customer VR

SNMP access capabilities for the customer VRs provide a logical IP connectivity and routing view. The physical information contained in the ifTable, and the Ethernet-like and FrDte Standard MIBs, is only visible through the management VR SNMP access.

Standard MIBs supported by virtual routers

The Standard MIBs supported on the customer VRs are a subset of the Standard MIBs supported on the management VR. The table, “Standard MIBs supported on management and customer virtual routers” (page 65), lists the Standard MIBs which are supported on the management and customer VRs.

Table 9
Standard MIBs supported on management and customer virtual routers

RFC	Name	Management VR	Customer VRs
RFC 1213	Management Information Base for Network Management of TCP/IP-based internets: MIB-II	X	X
RFC 1253	OSPF Version 2 Management Information Base	X	X
RFC 1315	Management Information Base for Frame Relay DTEs	X	
RFC 1354	IP Forwarding Table MIB	X	X
RFC 1471	The Definitions of Managed Objects for the Link Control Protocol of the Point-to-Point Protocol	X	X
RFC 1473	The Definitions of Managed Objects for the IP Network Control Protocol of the Point-to-Point Protocol	X	X
RFC 1573	Evolution of the Interfaces Group of MIB II	X	X
RFC 1643	Definitions of Managed Objects for the Ethernet-like Interface Types	X	
RFC 1657	Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2	X	X
RFC 1724	RIP Version 2 MIB Extension (see Note)	X	X
RFC 1748	IEEE 802.5 MIB using SMIv2	X	

(Sheet 1 of 2)

Table 9 (continued)
Standard MIBs supported on management and customer virtual routers

RFC	Name	Management VR	Customer VRs
RFC 1749	IEEE 802.5 Station Routing MIB using SMIv2	X	
RFC 1907	Management Information Base for Version 2 of the Simple Network Management Protocol	X	X
Note: No write support. Read only.			
(Sheet 2 of 2)			

Multiple SNMP agents

Each VR has its own SNMP agent process. The SNMP agent associated with a specific VR is activated when the SNMP component is provisioned for that VR. Therefore, only those VRs which have provisioned the SNMP component have SNMP agents.

The system group information for the customer VRs differs from the management VR. Depending on the type of VR, the following objects within the system group interface will contain different information:

- *sysDescr* is the textual description for the entity
  - — For the management VR, this value reflects the Passport switch.
  - — For the customer VR, this value reflects a Passport virtual router.
- *sysObjectID* is the vendor's identification of the entity
  - — For the management VR, this value reflects the Passport switch.
  - — For the customer VR, this value just reflects a Passport virtual router.
- *sysUpTime* is the length of time since the entity was re-initialized
  - — For the management VR, this value reflects the last time the Passport node was re-initialized.
  - — For the customer VR, this value reflects the time the VR was re-initialized.

For each SNMP agent, you can provision your own views. Similarly, for each SNMP agent you can provision your own community-related information. Each SNMP agent contains its own statistical counters.

The object resource table for each SNMP agent contains the same object resource entries. When a new MIB is loaded, each object resource table for all SNMP agents is updated with the object resource entry associated with that MIB.

To configure the SNMP Agents on the management and customer VRs, see "Configuring the Snmp component" (page 26).

Multiple virtual routers and the SNMP ifTable

The SNMP interface table (ifTable) for each VR contains the interface entries associated with its own VR. The only exception is the ifTable associated with the management VR.

The entries in the management VR ifTable include any interfaces associated with its own VR and all the non-VR related entries. The non-VR entries are layer 2 and layer 3 interfaces that are either Passport physical interfaces or Passport network services not associated with a VR.

The table, "Unrelated and specific VR SNMP ifTable entries" (page 67), lists the ifTable entries and indicates whether the interface is a non-VR related entity or related to a specific VR.

Table 10
Unrelated and specific VR SNMP ifTable entries

Entry type	Non-VR entries	Specific VR entries
AtmMpe		X
Btds	X	
DpnGate	X	
Frame Relay NNI	X	
Frame Relay to Atm	X	
Frame Relay UNI	X	
(Sheet 1 of 2)		

Table 10 (continued)
Unrelated and specific VR SNMP ifTable entries

Entry type	Non-VR entries	Specific VR entries
FrDte	X	
FrDteRemoteGroup		X
FrMux	X	
Htds	X	
Lan Application		X
Lp DS1	X	
Lp DS1 Chan	X	
Lp DS3	X	
Lp E1	X	
Lp E1 Chan	X	
Lp E3	X	
Lp Ethernet	X	
Lp V35	X	
Lp X21	X	
MPanl	X	
PointToPoint Protocol		X
Trunk	X	
Vs	X	
x25Dte	X	
x25DteRemoteGroup		X

(Sheet 2 of 2)

Restricting SNMP access to customer virtual routers

The internetworking service provider can restrict the SNMP access on customer VRs using the following capabilities:

- Restricting which SNMP Managers can access the customer VR by provisioning the IP addresses which are allowed within the *SNMP Manager* component.
- Restricting which SNMP requests are allowed on a customer VR by provisioning the *SNMP Manager* component privileges attribute.
- Restricting the access to ReadOnly using the SNMP *Community accessMode* attribute.
- Restricting which Standard MIBs are visible on the customer VRs using the SNMP view component.

For more information on how views can be used, see “SNMP Administrative model” (page 41).

Chapter 4
Passport Enterprise MIBs

SNMP Enterprise MIBs are vendor-specific MIBs. This chapter describes Passport Enterprise MIBs, and how Passport's proprietary data model (the Passport component model) is realized using SNMP MIBs, making the management of Passport via SNMP possible. The following topics are covered:

Content and format

Each Passport Enterprise MIB has a MIB name and is associated with a particular feature. When the feature is provisioned, the Passport MIBs associated with that feature are automatically installed into a global MIB tree on the Passport node.

For example, the DPRS feature has one associated Passport MIB named *Nortel-MsCarrier-MscPassport-DpnRoutingMIB* (*DpnRoutingMIB*). This MIB is loaded on the Passport node when the DPRS feature is provisioned.

Passport MIBs are supplied in two formats:

- SMIv1 format, based on RFC 1212 and RFC 1215 (required for SNMPv1 messages)
- SMIv2 format, based on RFC 1902 (required for SNMPv2c messages)

Refer to RFC 1908 for information on the difference between SMIv1 and SMIv2 syntax.

Each Passport MIB also contains:

- "MIB name" (page 72)
- "Imports clause" (page 72)
- "Module-identity statement" (page 74)
- "Management object definitions" (page 74)
- "Object-group and Agent-capabilities" (page 75)
- "End of the MIB" (page 76)

MIB name

All Passport MIBs begin with the assignment of a *moduleReference* (a MIB name) of the format:

Nortel-MsCarrier-MscPassport-MIBName

Where *MIBName* is replaced with a human-friendly reference name for the Passport MIB. A *MIBName* always ends with 'MIB'.

For example, the *FrameRelayUniMIB* begins with a *moduleReference* statement as follows:

```
Nortel-MsCarrier-MscPassport-FrameRelayUniMIB
DEFINITIONS ::= BEGIN
```

Imports clause

Immediately following the *moduleReference* statement is the *IMPORTS* clause. This clause imports, as required,

- macros and types from RFC 1212.

- standard textual conventions, defined in various RFCs, that are supported by the Passport Enterprise MIB.

  For your convenience, these have been repackaged into the *Nortel-MsCarrier-MscPassport-StandardTextualConventionsMIB*. In this way, you do not have to search for, acquire, and compile the actual source Standard MIBs that contain the standard textual conventions required by Passport Enterprise MIBs (see "Textual conventions" (page 229)).

- Passport textual conventions used (see "Textual conventions" (page 229)).
- OBJECT IDENTIFIER prefixes required for OBJECT IDENTIFIER assignments made in this module (see "SNMP object identifier usage" (page 221)).

See the following example of an IMPORTS clause.

```
IMPORTS

        OBJECT-TYPE
    FROM Nortel-MsCarrier-MscPassport-

        mscPassportMIBs,
        mscComponents
    FROM Nortel-MsCarrier-MscPassport-
     UsefulDefinitionsMIB

        Counter32,
        RowPointer,
        DisplayString,
        StorageType,
        RowStatus,
        InterfaceIndex,
        Gauge32,
        Integer32,
        Unsigned32
     FROM Nortel-MsCarrier-MscPassport-
      StandardTextualConventionsMIB

        AsciiString,
        Hex,
        NonReplicated,
        FixedPoint3,
        HexString,
```

```
        EnterpriseDateAndTime
        PassportCounter64
        Link
        DigitString
        Unsigned64
      FROM Nortel-MsCarrier-MscPassport-
       TextualConventionsMIB;
```

Passport Enterprise MIBs do not export any types or values; therefore, the *EXPORTS* clause is never included in a Passport MIB.

```
No EXPORTS clause
```

Module-identity statement

Contact and revision history is provided for each Passport Enterprise MIB with a *Module-identity* statement. For example, in the *FrameRelayUniMIB*, the *Module-identity* statement is specified as follows:

```
-- LAST-UPDATED "9909010000Z"
-- ORGANIZATION "Nortel Networks"
-- CONTACT-INFO "
--   Nortel Carrier Data Network Management
--
--   Postal:       P.O. Box 3511, Station C
--                 Ottawa, Ontario
--                 Canada K1Y 4H7
--
--   via the WEB:  http://www.nortelnetworks.com
--                 Select ‘Contact Us’ from the menu.
--   via phone:    1-800-4Nortel”
--
-- DESCRIPTION
--    "The module describing the Nortel MsCarrier
--     MscPassport FrameRelayUni Enterprise MIB."
098frameRelayUniMIB OBJECT IDENTIFER ::=
{ mscPassportMIBs 24 }
```

Management object definitions

The main body of the Passport MIBs includes MIB object definitions that model Passport components, groups, attributes, and alarms. Refer to “Mapping the component model to Passport Enterprise MIBs” (page 79) for more information on mapping Passport MIBs to the Passport management information.

Object-group and Agent-capabilities

The module concludes with an OBJECT-GROUP statement representing a group containing all variables defined in the module and an AGENT-CAPABILITIES statement for this version of this module. This AGENT-CAPABILITIES statement also describes support for any Standard MIB modules loaded in conjunction with the Enterprise MIB module.

Note: The software version numbers for the software installed on your system may differ from those shown in the following example.

```
-- Object-Group Statement:
-- A list of all current accessible leaf objects.

--OBJECTS { mscFrUniDlciVcPriority,
--          mscFrUniDlciDiscardedSegFrm,
--          mscFrUniLtsPatDefaultHeaderLength,
--          mscFrUniDnaHgMHgAddrComponentName,
--          mscFrUniDlciLbLocalBecnFrm,
--          ...
--          mscFrUniFrmToIfByQueueValue }
--STATUS current
--DESCRIPTION
--  “A list of all current accessible leaf objects.”

frameRelayUniGroup OBJECT IDENTIFIER
::= { frameRelayUniMIB 1}
frameRelayUniGroupCA OBJECT IDENTIFIER
::= { frameRelayUniGroup 1}
frameRelayUniGroupCA02 OBJECT IDENTIFIER
::= { frameRelayUniGroupCA 3}
frameRelayUniGroupCA02DevelopmentLoad OBJECT
IDENTIFIER ::= { frameRelayUniGroupCA02 1}
frameRelayUniGroupCA0211 OBJECT IDENTIFIER
::= { frameRelayUniGroupCA02DevelopmentLoad 11}
frameRelayUniGroupCA0211A OBJECT IDENTIFIER
::= { frameRelayUniGroupCA0211 1}


-- Agent-Capabilities Statement:
-- PRODUCT-RELEASE “MscPassport Release 1.2
                   FrameRelayUni."
-- STATUS          current
```

```
-- DESCRIPTION     “Carrier Passport
                   FrameRelayUni MIB:
--                 MIB Version CA0211A, Software
--                 Version CA0211A.”
-- SUPPORTS        Nortel-MsCarrier-MscPassport-
                   FrameRelayUniMIB
--    INCLUDES     { frameRelayUniGroupCA0211A }

frameRelayUniCapabilities OBJECT IDENTIFIER
::= { frameRelayUniMIB 3 }
frameRelayUniCapabilitiesCA OBJECT IDENTIFIER
::= {frameRelayUniCapabilities 1}
frameRelayUniCapabilitiesCA02 OBJECT IDENTIFIER
::= {frameRelayUniCapabilitiesCA 3}
frameRelayUniCapabilitiesCA02DevelopmentLoad OBJECT
IDENTIFIER ::= {frameRelayUniCapabilitiesCA02 1}
frameRelayUniCapabilitiesCA0211 OBJECT IDENTIFIER
::= {frameRelayUniCapabilitiesCA02DevelopmentLoad 11}
frameRelayUniCapabilitiesCA0211A OBJECT IDENTIFIER
::= {frameRelayUniCapabilitiesCA0211 1}
```

End of the MIB

All MIBs have an END statement, as follows:

```
END
```

Component model

Passport defines a proprietary data model called the component model, which is used to specify the various resources that can be managed. The component model structures Passport management information using the concepts of a component and a component hierarchy.

A component represents the management view of a real resource. A component may represent a physical resource such as a card, a logical resource such as a *Framer*, or even a collection of information concerning a resource. For example, a *Dna* component provides information concerning a frame relay UNI service.

Components make a network resource manageable by

- providing a name for the resource (the component name).

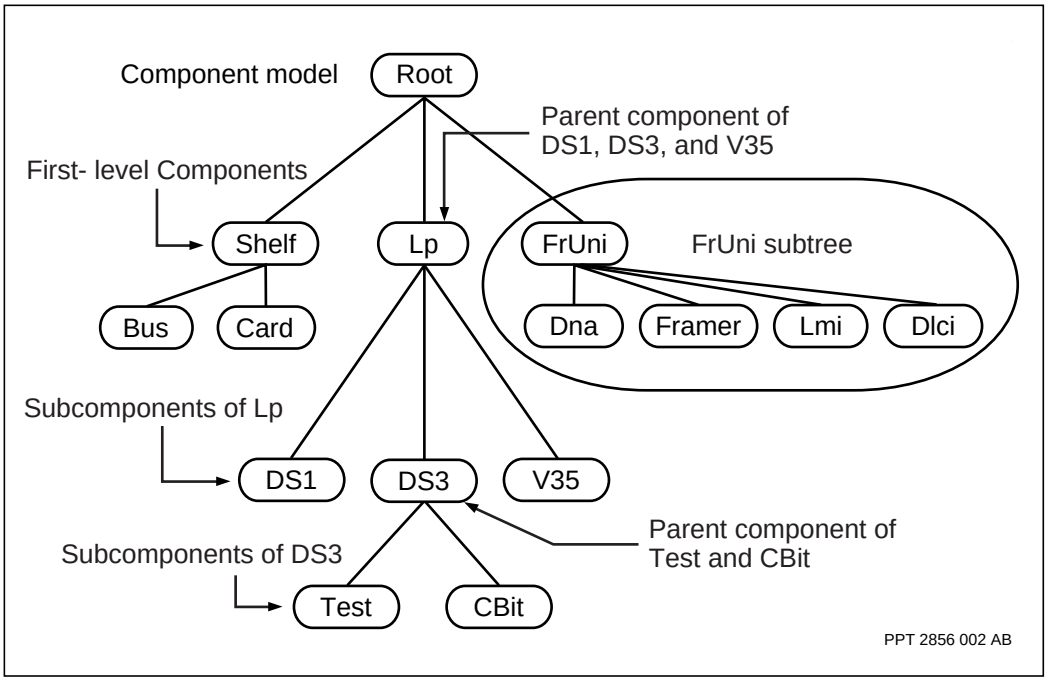
- providing information concerning the resource through groups of attributes.
- providing commands to perform specific actions against a resource

Component hierarchy and subcomponents

Components are arranged into a tree-like structure or hierarchy that defines the relationship between the individual components. This is shown in the figure “Component hierarchy tree” (page 77). Through this structure, a containment relationship is developed between each of the components that combine to form the Passport system.

The component hierarchy is comprised of a root node followed by the first level of components called the top level components. For example the *Lp* component contains the components *DS1, DS3* and *V35*. The subcomponent *DS3* contains the components *Test* and *CBit*.

Figure 15
Component hierarchy tree



By their relative positions in the hierarchy, components may have a parent component and zero to many subordinate components defined. A component which is subordinate to another component is called a subcomponent. For example, the *FrUni* component has four subcomponents: *Dna*, *Framer*, *Lmi*, and *Dlci*.

Subcomponents have all the normal characteristics of components and they have a parent component.

Passport component class and instance

There are many different kinds of network resources, and so there are many different kinds of components: components are organized based on the concepts of component class and instance.

Components are organized by component class for the purpose of specification and for naming. The component class specifies a path in the component hierarchy starting from a top level component and ending with a subcomponent. For example, *FrUni Dlci* is a component class just as *Lp DS3 Test* is a component class.

A component instance refers to a particular instance of a component on a particular node. An example of a component instance for the component class *FrUni Dlci* is *FrUni/1 Dlci/19*. In this example, the component instance refers to the *FrUni* component with instance 1, and the *Dlci* component with instance 19.

Attributes and attribute groups

Information about a component is represented by attributes organized into groups. Each attribute describes a particular characteristic of the component. There are two kinds of attributes: operational and provisionable.

Operational attributes represent operational information about a component, such as state and statistical information. This information is used for monitoring the operation of Passport and its services. Values of operational attributes can be set if the attribute is defined to allow writes, but the values are not retained across system restarts. The values of operational attributes may change at any time independent of operator intervention.

Provisionable attributes define a component's behavior. This information is used to control the operation of Passport and its services. Values of provisionable attributes can be configured (provisioned) so that Passport subsystems or services perform or behave in a certain manner. These values are nonvolatile or persistent; that is, they are retained across system restarts. Values of provisionable attributes change only as a result of operator intervention.

A group can contain attributes of only one kind. Components can contain both operational and provisionable attribute groups.

Mapping the component model to Passport Enterprise MIBs

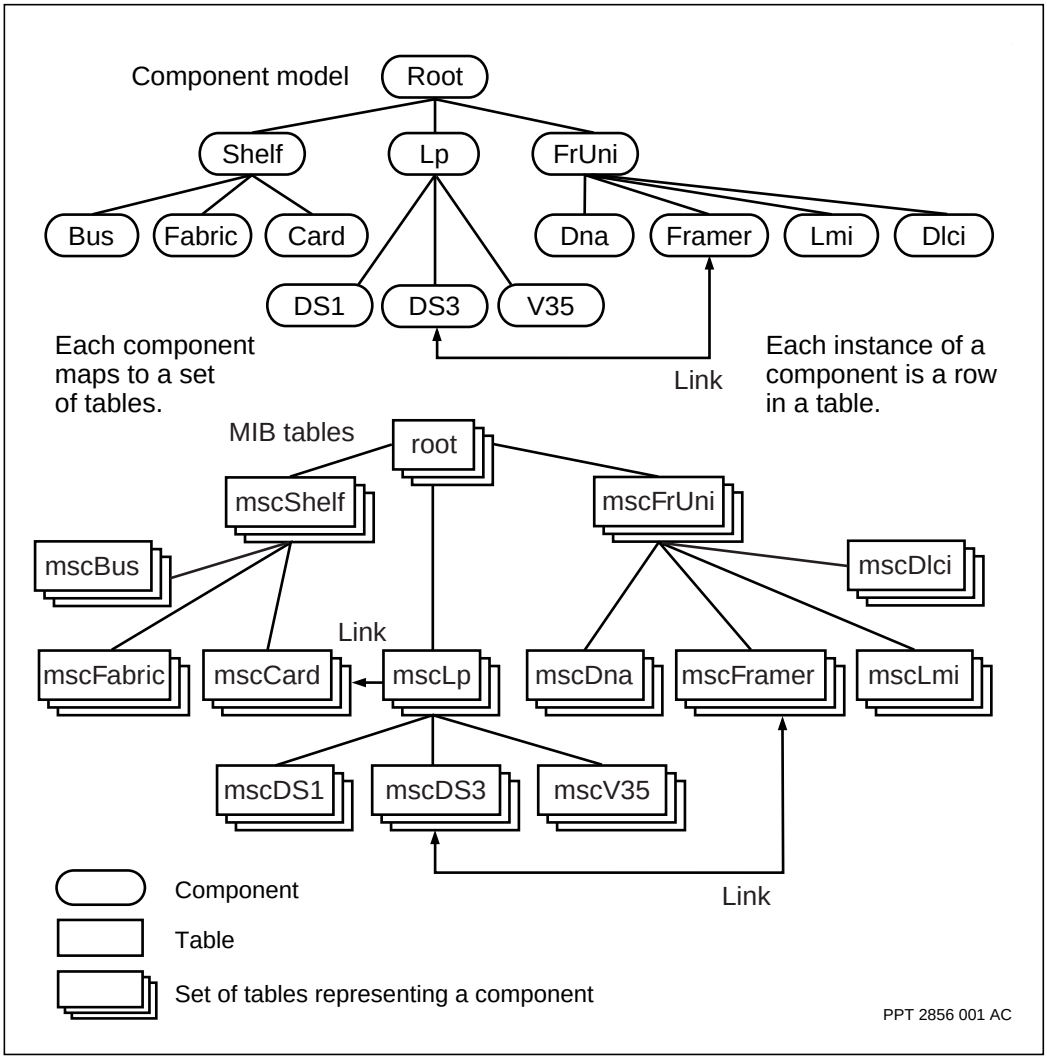
Passport Enterprise MIBs reflect the Passport component model, as follows:

- Component classes are represented in Passport Enterprise MIBs as a set of SNMP tables.
- A Passport attribute group is represented by an SNMP table.
- A Passport attribute is represented by an SNMP variable. Structured Passport attributes are mapped onto SNMP tables.
- Passport data types are mapped onto SNMP base data types or Passport specific textual conventions.
- The hierarchy of the SNMP tables is identical to the Passport component model hierarchy.

Non-replicated components are represented as an SNMP table, even though there is only one instance of the component. In this instance, the component is represented as an SNMP table with one instance; therefore, there will never be scalar SNMP variables defined in the Passport Enterprise MIBs.

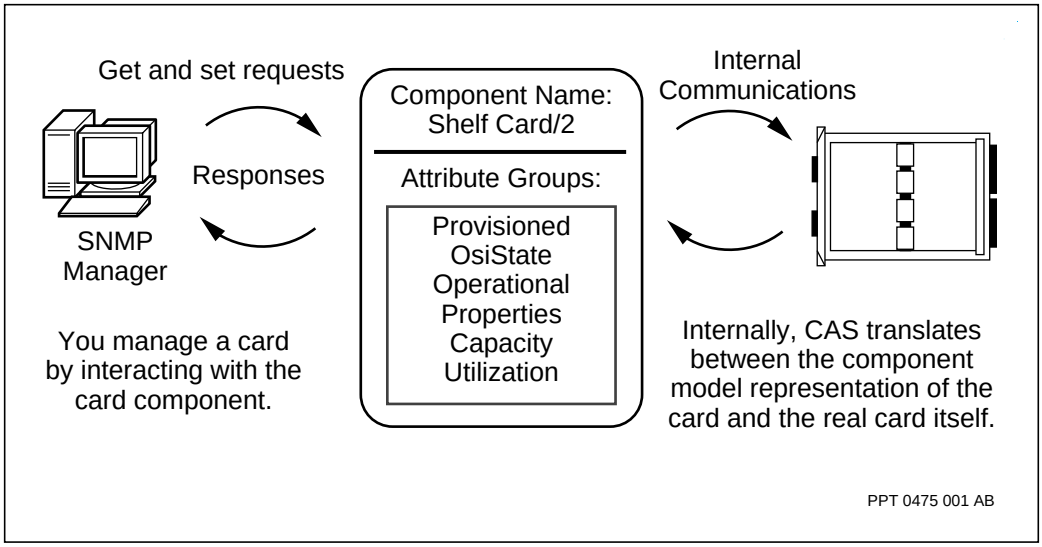
The mapping of the component model onto SNMP MIB tables is illustrated in the figure "Component mapping example" (page 80).

Figure 16
Component mapping example



This means the advantages of the component model are available through SNMP as depicted in the figure “The Card component” (page 81). SNMP requests can be operated on the component model via Passport Enterprise MIBs.

Figure 17
The Card component



The hierarchical arrangement of components into a tree is reflected in the structure of the Passport Enterprise MIBs. This structure has a number of advantages, which include

- creating a deeper tree, making the structure easier to browse.
- allowing SNMP *getNext* sweeps to target specific parts of the component hierarchy.
- aligning with SNMP hierarchal-based access control schemes.
- allowing SNMP *getBulk* requests for efficient retrieval of large amounts of information.

Naming convention

SNMP object names and object identifiers are used to reference each component class, component, subcomponent, group, attribute, and structure attribute in the component model.

Passport SNMP object names prefix each component class, component, subcomponent, group, attribute, and structure attribute in the component model with ‘msc’. For example, the *FrUni* component has an SNMP object name of *mscFrUni*.

Passport SNMP object identifiers are composed of a series of integer values separated by a period ‘.’. For example, the object identifier ‘.1.3.6.1.4.1.562.36.2.1.71.5’ is used to identify the *FrUni Dlci* component class. An arc numbering plan is used to maximize the long-term user-friendliness of the Passport Enterprise MIBs.

- Top-level components are numbered relative to the *{ mscPassport(2) mscComponents(1) }* arc, beginning at arc *11*. This allows space for special components such as *mscCasRoot(1)*, *mscTransactions(2)*, *mscPassportTraps(3)*, and *mscExtensions(4)*
- When *RowStatusTable* is subordinate to a component, it is always assigned arc *1*.
- Subcomponent assignments follow arc 1, as it is reserved for the *RowStatusTable*.
- Attribute group assignments follow at arcs 10 or 100.
- Structured attribute tables are assigned arcs beginning at *200*. These assignments are global; no two structured attribute tables have the same arcs. See “Structured attribute definition” (page 89) for more information.

Component classes in the Enterprise MIB

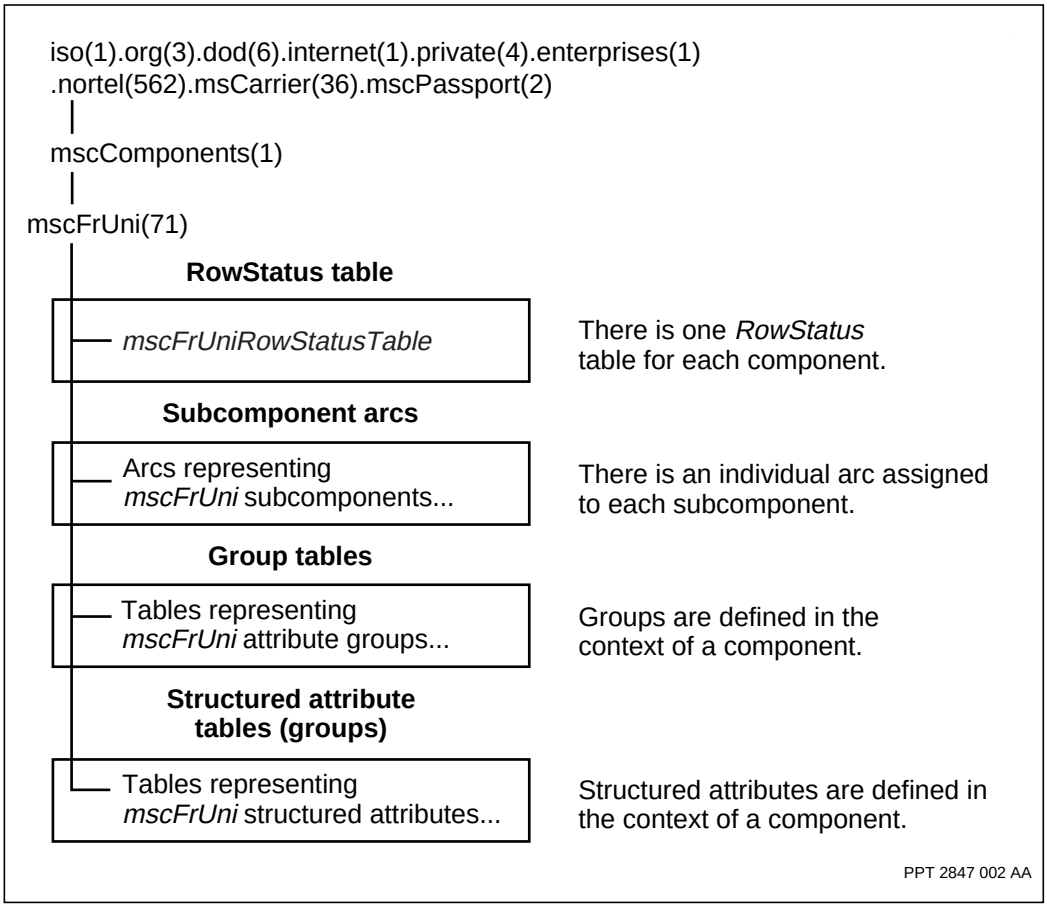
Component classes are defined in the Passport Enterprise MIBs by

- assigning an SNMP object identifier to the component class. All information concerning components of this type, including all information concerning its subcomponents, can be found in the subtree rooted at this object identifier.
- providing an SNMP *RowStatusTable* for the component. This table provides important general information concerning the component, and allows addition and deletion of the component.
- defining SNMP tables for the component’s attribute groups and structured attributes.

- subcomponents are defined in the same way that components are defined; by assigning an OBJECT IDENTIFIER to the subcomponent, providing a *RowStatusTable*, defining tables for its attribute groups and structured attributes and defining its subcomponents.

All components have exactly the same structure in the Passport Enterprise MIBs. This structure is illustrated with the *mscFrUni* subtree in the figure "Generic MIB structure using mscFrUni as an example" (page 83).

Figure 18
Generic MIB structure using mscFrUni as an example



Component definition

Within a Passport Enterprise MIB, the definition of a component type is as follows:

```
-- FrameRelayUni/n
-- This component defines an instance of the Frame
-- Relay UNI service. The component instance value n
-- is a unique positive integer within a module. Framer
-- is created whenever the FrUni is added. Users who
-- need VirtualFramer should delete the Framer first,
-- then add the VirtualFramer.
mscFrUni OBJECT IDENTIFIER ::= { mscComponents 71 }
```

Information concerning the *FrUni* component can be found in the Enterprise MIB below the arc *mscFrUni(71)* or object identifier 1.3.6.1.4.1.562.36.2.1.71. Currently, not all tables related to dynamic unregistered replicated components are supported.

RowStatus table

All components have a *RowStatusTable* that is always assigned arc 1 below the component arc. The *RowStatusTable* contains important information concerning the component. When a row is added to the *RowStatusTable*, this implies that a new component instance is created. Therefore, a new row is also added for each related attribute group table and structured attribute table of

the newly created component instance. The *RowStatusTable* also contains the index variables for the component. These same variables are used to index all related attribute group tables and structured attribute tables.

- the *RowStatus* variable, always assigned arc 1 beneath the RowStatusTableEntry, allows for component addition and deletion. The only valid values returned on a GET of RowStatus are notInService or active.

  A value of notInService is returned if the component has not been activated due to the lp being down, or an activation error (such as resource unavailable). When in this state, a GET on variables in operational tables for this component will return zero/null/invalid values.

  A value of active is returned if the component has been activated by the local management system.

  The exceptions to this include *mscShelfCardRowStatus* and *mscLpRowStatus*. The *mscShelfCardRowStatus* component is always active regardless of whether the card is inserted or not. Use the *mscShelfCardInsertedCardType* or *mscShelfCardAvailabilityStatus* to determine if the card is inserted. The *mscLpRowStatus* is always active. Use *mscLpActiveCard* to determine if it is up and active.

  Note: The object identifier of this variable may be used as a RowPointer as it implicitly refers to the component name.

- *ComponentName* variable, always assigned arc *2* beneath the *RowStatusTableEntry*, provides an ASCII name for the component which may be used via the ASCII console
- where applicable, an *IfIndex* variable, always assigned arc *3* beneath the *RowStatusTableEntry*, provides an *IfIndex* name form for the component which may be used to correlate information available from Standard MIBs
- where applicable, a *StorageType* variable, always assigned arc *4* beneath the *RowStatusTableEntry*, provides a mechanism for converting dynamic components to operational components.

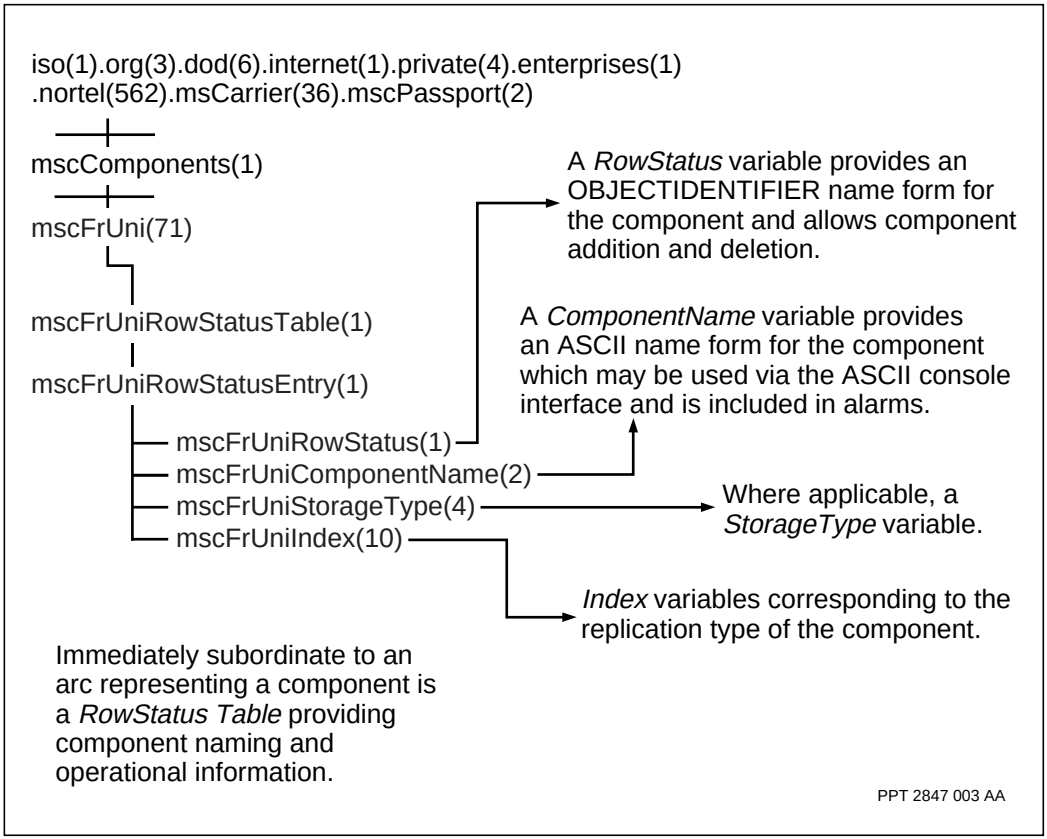
- Index variables corresponding to the replication type of the component (for example, *mscFrUniIndex*). These variables will begin at arc 10 beneath the *RowStatusTableEntry*. In some cases more than one index variable may be used.

The reference name of the *RowStatusTable* for a component is composed of the component type name suffixed by the string *RowStatusTable.* An example is *mscFrUniRowStatusTable* in the figure "The RowStatus table" (page 87). The names of the other variables in the *RowStatusTable* are similarly formed.

Note 1: When a component instance is named in SNMP, for example, as a value of a *Link* attribute, the SNMP OBJECT IDENTIFIER value references the component using the *RowStatus* variable of the component's *RowStatusTable*. For example, the component *FrUni/21* has the SNMP OBJECT IDENTIFIER name *mscFrUniRowStatus.21*.

Note 2: The *RowStatusTable* is only visible through SNMP. It is not visible from the console or through a telnet session.

Figure 19
The RowStatus table



Attribute and attribute group definition

Attribute groups are defined as SNMP MIB tables. Individual attributes are defined as the variables comprising the columns of the table.

There are three important parts to the definition of an attribute group:

- the table definition, which provides a description of the group
- the entry definition, which provides the index information for the table
- the variable definitions, which provide the attribute definitions for the attributes of the group

Attribute group definition

Passport attribute groups are defined as SNMP tables. These tables are typically assigned arcs beginning at 10 and immediately follow the *RowStatusTable* definition as described in MIB modules. For example:

```
mscVsFramerSignalTable OBJECT-TYPE
SYNTAX SEQUENCE OF MscVsFramerSignalEntry
ACCESS not-accessible
STATUS current
DESCRIPTION
“This group contains attributes which define the
means for transporting channel associated signalling
through the network.”
::= { mscVsFramer 13 }
```

A description of the semantics of the group is supplied with the table definition.

The table entry definition provides index information:

```
mscVsFramerSignalEntry OBJECT-TYPE
SYNTAX MscVsFramerSignalEntry
ACCESS not-accessible
STATUS current
DESCRIPTION
“An entry in the vsFramerSignalTable.”
INDEX { mscVsIndex, mscVsFramerIndex }
::= { mscVsFramerSignalTable 1 }
```

These tables always augment the *RowStatusTable* for the component. This reflects the fact that the index information is the same for attribute group tables and the *RowStatusTable*.

> ***Note:*** In SNMP modules, the index clause for attribute group tables is the same as that for the *RowStatusTable*.

Attribute definition

Passport attributes are defined as SNMP MIB variables. The variable definition provides the syntax of the variable, its accessibility, status, and a description of its semantics. For example:

```
mscVsFramerTransmitBusyYellow OBJECT-TYPE
SYNTAX INTEGER  {
no(0),
yes(1) }
ACCESS read-write
STATUS current
DESCRIPTION
“When this attribute is set to yes and the DS1 or E1
line is set to Channel Associated Signaling (CAS)
mode, a ‘busy-out’ signaling condition is transmitted
onto the voice timeslot associated with this Framer
when its path across the network is down. The busy-out
code transmitted is given by the seizeCode attribute.
If transmitBusyYellow is set to yes on a CAS-based
interface, the seizeCode attribute must be set. When
transmitBusy is set to yes and the DS1 or E1 line is
set to Common Channel Signaling (CCS) mode, a yellow
alarm condition is transmitted onto the DS1/E1 line
when the path across the network associated with this
Framer is down.
Note that in CCS mode, a yellow alarm condition is
transmitted when the path is down for any of the
Framers with this attribute set to yes.”
DEFVAL     { no }
::= { mscVsFramerSignalEntry 1 }
```

Structured attribute definition

Passport structured attributes are defined as SNMP tables. Passport defines four kinds of structured attributes:

- vector
- array
- list
- replicated

Vectors and arrays are structured attributes consisting of a fixed number of values of a particular base type. Individual values are indexed and can be addressed using an index value.

> ***Note:*** The *set of* type is considered as a structured type from the text interface, but it is mapped onto the SNMP type OCTET STRING.

Lists are structured types consisting of a number of values of a particular base type. Individual values are not indexed and cannot be addressed using an index value. The tables representing these attributes are indexed by the values themselves.

Replicated attributes are structured attributes consisting of a variable number of values of a particular base type. Individual values are indexed and can be addressed using an index value.

Lists and replicated attributes support adding and deleting of particular values. A *RowStatus* variable, provided in tables representing attributes of these types, is used for deleting values. This variable is implemented as *write-only* and cannot be read. For more information on adding and deleting values, see “Provisioning using SNMP set request” (page 213).

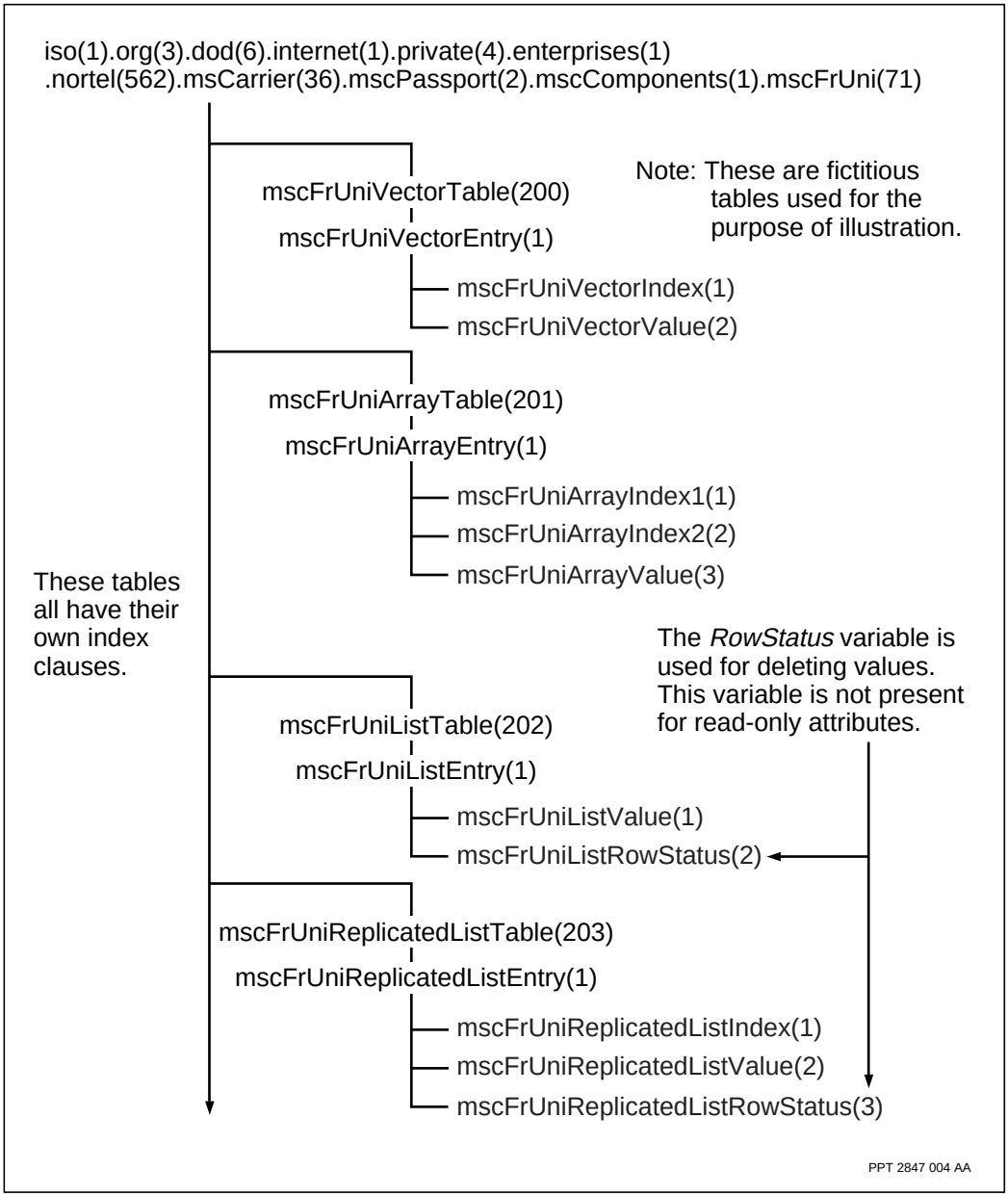
> ***Note:*** Lists and replicated attributes that are *read-only* do not have *RowStatus* variables.

Tables representing structured attributes are assigned arcs at the same level as arcs representing attribute groups. In other words, structured attributes are promoted to the group level. In MIB modules, these definitions follow the definitions of the tables representing the component’s attribute groups.

The figure, “MIB tree structure for structured attributes” (page 91), shows the MIB tree structure for four fictitious structured attribute tables:

- *mscFrUniVectorTable*
- *mscFrUniArrayTable*
- *mscFrUniListTable*
- *mscFrUniReplicatedListTable*

Figure 20
MIB tree structure for structured attributes



Example of structured attribute of type List

The *mscShelfCardConfiguredLPsTable* provides an example of a structured attribute definition: a List. This attribute is defined in the *BaseShelfMIB*.

As with attribute group tables, there are three parts to the definition of a structured attribute table:

- the table definition, which provides a description of the structured attribute
- the entry definition, which provides the index information for the table
- the variable definitions, which provide the value and index definitions for the table

The table definition provides a description of the attribute and assigns an OBJECT IDENTIFIER value subordinate to *shelfCard*, the component to which this attribute belongs. The *shelfCardConfiguredLPsTable* is the only structured attribute in the Enterprise MIB whose OBJECT IDENTIFER arc is *243*.

```
mscShelfCardConfiguredLPsTable OBJECT-TYPE
SYNTAX SEQUENCE OF MscShelfCardConfiguredLPsEntry
ACCESS not-accessible
STATUS current
DESCRIPTION
“This is a read-only attribute that shows which LPs
are currently configured to run on this card (it is
the opposite side of the LP component’s mainCard and
spareCard link attributes). Note that there is only
one case where more than one LP can be configured to
run on a card. This is when the card is configured as
an N+1 spare. If this is not the case, the card
is configured as a main card or as a 1+1 spare, only
one LP can appear in this list.”
::= { mscShelfCard 243 }
```

The next part of the definition is the entry. It provides the index information.

```
mscShelfCardConfiguredLPsEntry OBJECT-TYPE
SYNTAX MscShelfCardConfiguredLPsEntry
ACCESS not-accessible
STATUS current
```

```
DESCRIPTION
"An entry in the shelfCardConfiguredLPsTable."
::= { mscShelfCardConfiguredLPsTable 1 }
```

The indices include those used for the *RowStatusTable* of the component containing the attribute, plus an additional index which is specific to the structured attribute (*mscShelfIndex*, *mscShelfCardIndex* + *mscShelfCardConfiguredLPsValue*).

A variable definition is provided for the list values:

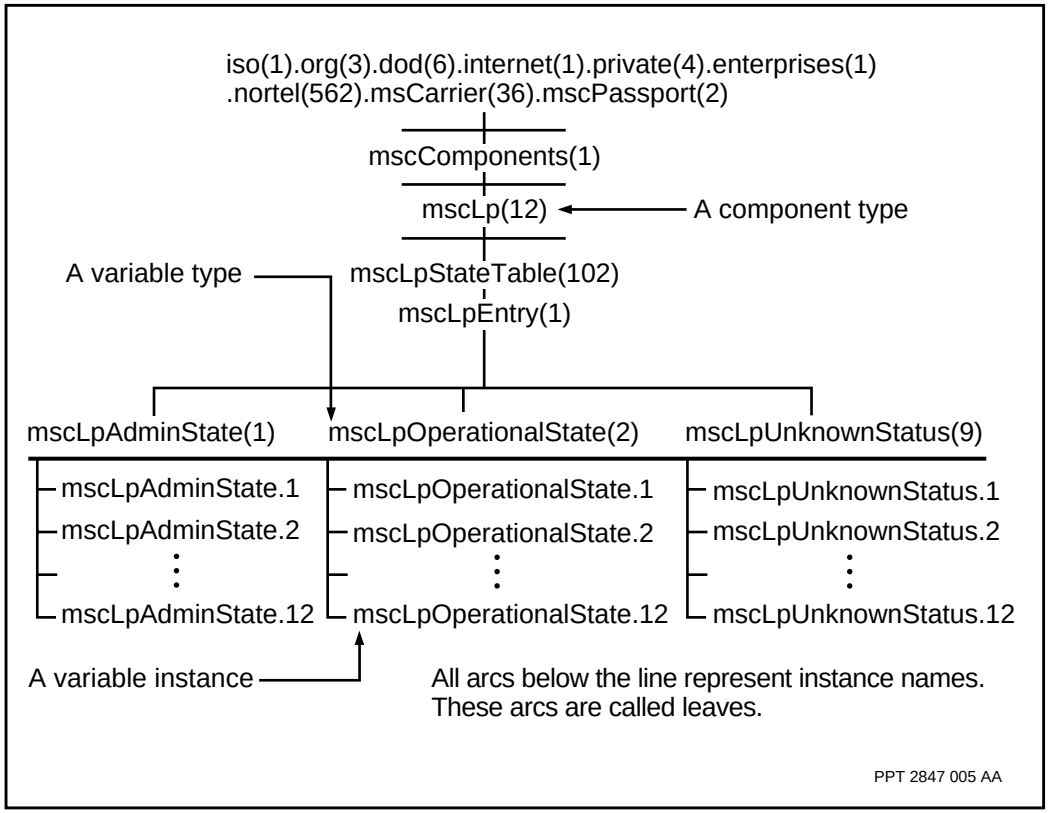
```
mscShelfCardConfiguredLPsValue OBJECT-TYPE
SYNTAX link
ACCESS read-only
STATUS current
DESCRIPTION
This variable represents both the value and the index
for the shelfCardConfiguredLPsTable."
::= { mscShelfCardConfiguredLPsEntry 1 }
```

As the variable *mscShelfCardConfiguredLPsValue* is read-only, there is no *RowStatus* variable.

How Passport Enterprise MIB tables are indexed

SNMP variables are referenced with an object identifier. Each object identifier is comprised of a variable type portion followed by an instance portion. Variable type and instance are illustrated in the figure, "Variable type and instance" (page 94), which presents the *mscLpStateTable* for a system with 12 logical processors.

Figure 21
Variable type and instance



Attributes

Attributes are grouped together in attribute groups, which are represented as SNMP tables under their respective component class. Each instance of an attribute is referenced by the instance of the component class that contains the attribute.

For example, the Passport attribute *elapsedTimeTillNow* under the *FrUni Dlci Vc* component class with a *FrUni* instance of 1 and a *Dlci* instance of 16 is referenced in the component model as:

frUni/1 Dlci/16 Vc ElapsedTimeTillNow

The same attribute is referenced with the SNMP variable OBJECT IDENTIFIER:
mscFrUniDlciVcElapsedTimeTillNow.1.16.1
(1.3.6.1.4.1.562.36.2.1.71.5.3.11.1.2.1.16.1)

where the variable type is:
mscFrUniDlciVcElapsedTimeTillNow (*1.3.6.1.4.1.562.36.2.1.71.5.3.11.1.2)*

and the instance is:
1.16.1

> ***Note:*** Unreplicated components such as the *Vc* component in this example takes an index value of 1.

Structured attributes

A Passport structured attribute is also represented as an SNMP table under its respective component class. These attributes are indexed by the instance of the component class that contains that attribute and the instance value of the structured attribute. Variables are defined in each structured attribute table representing the indices of the structured attribute.

The following is an example of a fictitious structured attribute, used for this example only.

The Passport structured attribute, *vectorAttribute*, under the *FrUni Dlci Vc* component class with a *FrUni* instance of 1 and a *Dlci* instance of 16 is referenced in the component model as:

frUni/1 Dlci/16 Vc vectorAttribute

where the attribute *vectorAttribute* has an arc value of:
251

> ***Note:*** The Passport Enterprise MIB allows elements of structured attributes to be referenced individually.

The ninth element of the *vectorAttribute* attribute is represented by the SNMP variable OBJECT IDENTIFIER as:
mscFrUniDlciVcVectorAttribute.1.16.1.9
(1.3.6.1.4.1.562.36.2.1.71.5.3.11.1.251.1.16.1.9)

where the variable type is:
mscFrUniDlciVcVectorAttribute (*1.3.6.1.4.1.562.36.2.1.71.5.3.11.1.251)*

and the instance is:
1.16.1.9

> ***Note:*** Unreplicated components such as the *Vc* component in this example takes an index value of 1.

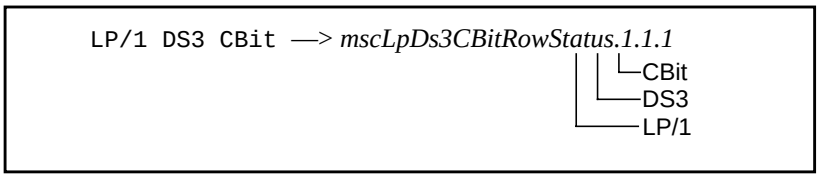
Link type attributes

There are attributes that are defined in Passport that reference other component class instances. An attribute of type *Link* contains a reference to a component class instance such as *FrUni/1* or *Lp/1 DS3*. SNMP defines a textual convention called the *RowPointer* for this purpose. A *RowPointer* represents a row, with its value being an OBJECT IDENTIFER. The value of the OBJECT IDENTIFER is used to represent the component class instance.

For example, *ifIndex.3* is the name of the third row in the *ifTable*.

In the Passport Enterprise MIB, the object identifier of the *RowStatus* variable in the *RowStatusTable* provides a machine-processable component name of type *RowPointer*. An example is shown in the figure "Machine-processable component name" (page 96).

Figure 22
Machine-processable component name



The *ComponentName* variable in the *RowStatusTable* provides an ASCII type name *mscLpDs3CBitComponentName.1.1.1* —> LP/1 DS3 CBit.

Detailed example of the FrUni component

The figure, "Partial object identifier tree for the FrUni component" (page 99), illustrates a partial representation of the Frame Relay UNI component hierarchy in the *FrameRelayUniMIB*:

- *FrUni* is a first level component and so has a SNMP arc directly subordinate to the *mscComponents* arc
- immediately subordinate to the *mscFrUni* arc is the *RowStatusTable*
- arcs representing subcomponents of the *FrUni* component (for example, *mscFrUniDna, mscFrUniFramer, mscFrUniLmi* and *mscFrUniDlci*) are assigned the next available arcs
- table definitions corresponding to the component's attribute groups follow (for example, *mscFrUniCidDataTable, mscFrUniStateTable, mscFrUniStatsTable,* and *mscFrUniProvTable*). These tables augment the *RowStatusTable*.

In the MIB, there is one and only one of each of the tables shown. For example, there is only one *mscFrUniStatisticsTable* for a particular node. This table includes rows for all *FrUni* components existing on the node. This table is a virtual table, realized in total by the collection of *FrUni* components. As *FrUni* components are first level components having an INTEGER-valued replication type, all tables within the MIB representing these components, and their subcomponents, are indexed by the INTEGER-valued variable *mscFrUniIndex*.

The figure, "Partial object identifier tree for the FrUni component" (page 99), illustrates the SNMP MIB view of the *FrUni* component model. It includes the naming of some of the variables contained within individual tables for the *mscFrUniStatsTable* and *mscFrUniDlciVcIntdTable*:

- The instance ID for the *FrUni* component appears as an index, *mscFrUniIndex*, for all tables for this hierarchy.
- For replicated subcomponents, the instance ID of the subcomponent also appears as an index to the table. For example, *mscFrUniDlciIndex* is used as an index into the *mscFrUniDlciVcIntdTable*. This applies recursively where there are subcomponents of subcomponents.

- For nonreplicated subcomponents, a special index of type *NonReplicated* is used. For example, the *Fr Uni Dlci Vc* subcomponent is a nonreplicated subcomponent of the *FrUni Dlci* component, so an index *mscFrUniDlciVcIndex* of type *NonReplicated* is required for the contained tables (this always has the value 1).

The recursive nature of the overall mapping is illustrated by the *mscFrUniDlciVcIntdTable*: a *mscFrUniDlciVc* is a nonreplicated subcomponent of the *FrUniDlci* component, which in turn is a replicated subcomponent of the *FrUni* component, itself a replicated component. This means that all *mscFrUniDlciVc* tables, such as the *mscFrUniDlciVcIntdTable*, have at least three indices: *mscFrUniIndex*, *mscFrUniDlciIndex* and *mscFrUniDlciVcIndex*. An index variable is defined once, but may be used in many tables.

Figure 23
Partial object identifier tree for the FrUni component

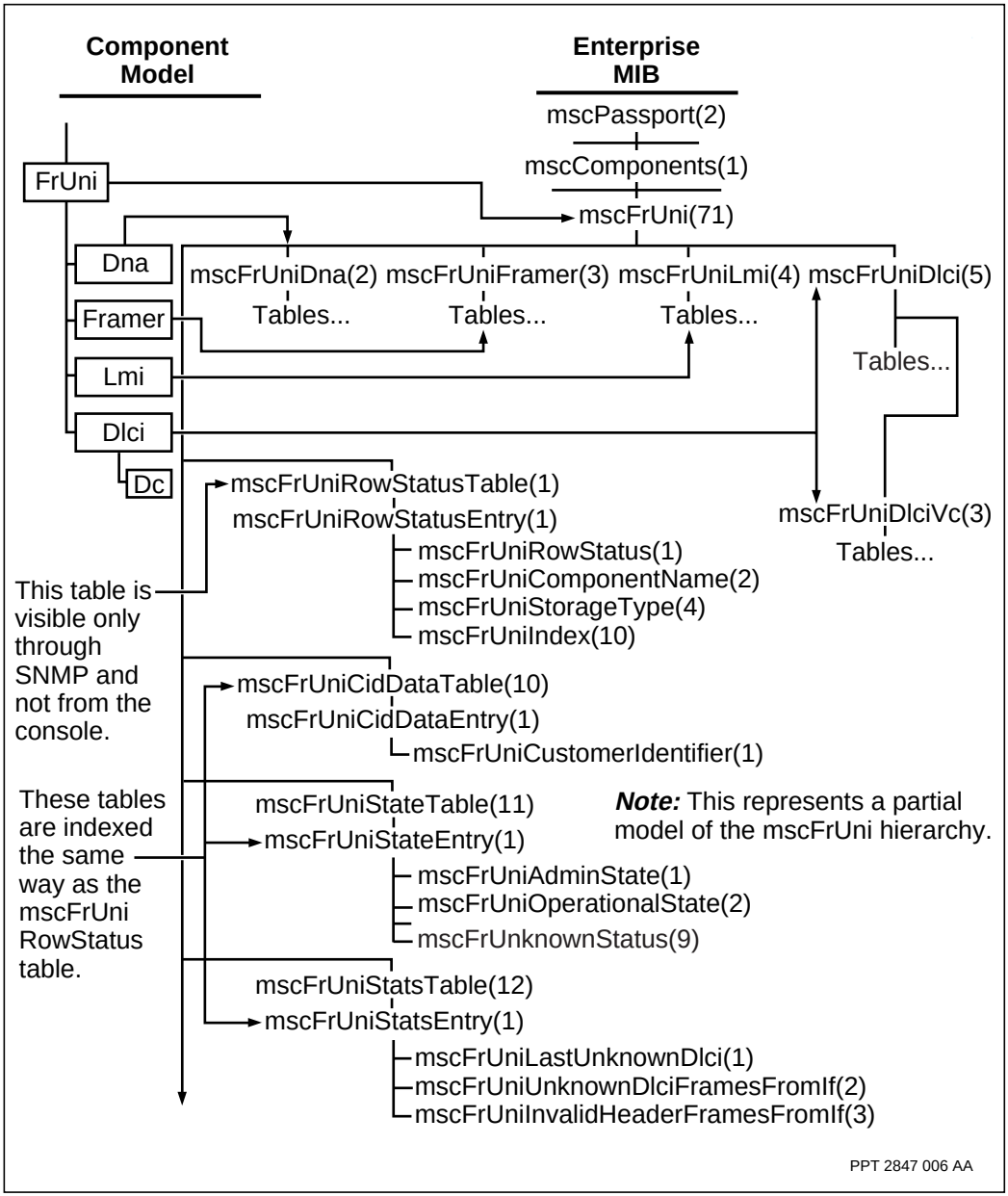
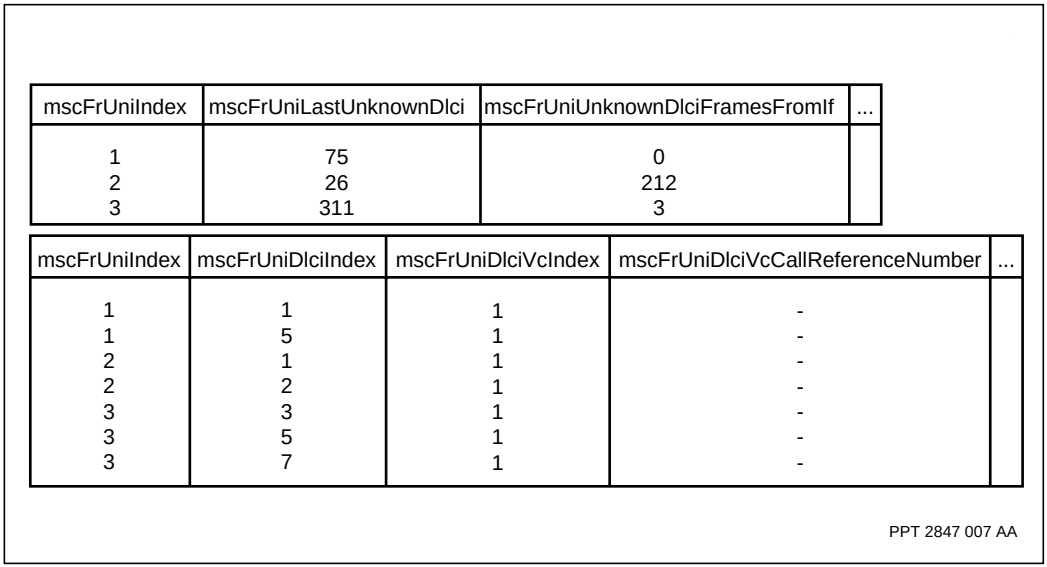


Figure 24
Some examples of Passport mscFrUni tables



mscFrUniIndex	mscFrUniLastUnknownDlci	mscFrUniUnknownDlciFramesFromIf	...
1	75	0	
2	26	212	
3	311	3	

mscFrUniIndex	mscFrUniDlciIndex	mscFrUniDlciVcIndex	mscFrUniDlciVcCallReferenceNumber	...
1	1	1	-	
1	5	1	-	
2	1	1	-	
2	2	1	-	
3	3	1	-	
3	5	1	-	
3	7	1	-	

PPT 2847 007 AA

Mapping Passport data types to SNMP data types

An attribute value is defined as a Passport component description language (CDL) data type. In the Enterprise MIB, each attribute value's data type is mapped to a standard SNMP base data type, a standard SNMP textual convention or a Passport specific textual convention.

- SNMP base types are defined in RFC 1155.
- SNMP generic textual conventions are defined in RFC 1903.
- Subsets of various Standard SNMP MIBs used by Passport have been repackaged into an Enterprise MIB, Passport MIB Module Standard Textual Conventions. This set includes textual conventions and base types from the following Standard MIBs:
  - — RFC 1155-SMI
  - — RFC 1253
  - — RFC 1315
  - — RFC 1382

  - — RFC 1493
  - — RFC 1512
  - — RFC 1573
  - — RFC 1903
- Passport-specific textual conventions are defined in the Passport MIB Module Textual Conventions

The Passport specific MIBs can be referenced through the Software Distribution Site (SDS).

The table, “Mapping of Passport CDL data types to Passport Enterprise MIB data types” (page 101), provides a mapping of Passport component data types to SNMP types for attributes of components.

Table 11
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
BCD	DigitString	OCTET STRING (SIZE (0..1024))	Represents a string of decimal digits, that is 0..9. One digit encoded in Ascii per octet.
BITSET		OCTET STRING (SIZE (1|2|3|4))	Represents a set of bits. Eight bits encoded per octet. Bit 0 of the first byte is the most significant bit.
COMPONENT_ NAME	RowPointer	OBJECT IDENTIFIER	RowPointer is used to signal that the object identifier represents a component name and may be translated to a component name.
DECIMAL		Gauge32	Represents an unsigned integer with its value fluctuating between a lower and upper bound.
		Counter32	Represents an unsigned integer that is increasing in value.
(Sheet 1 of 6)			

Table 11 (continued)
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
Enumerated Types		INTEGER	Represents an enumerated value encoded as an integer.
FIXED_POINT	FixedPoint1	Gauge32	Represents a fixed point value with one decimal place. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 12345678.9
	FixedPoint2	Gauge32	Represents a fixed point value with two decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 1234567.89
	FixedPoint3	Gauge32	Represents a fixed point value with three decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 123456.789
	FixedPoint4	Gauge32	Represents a fixed point value with four decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 12345.6789
	FixedPoint5	Gauge32	Represents a fixed point value with five decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 1234.56789
	FixedPoint6	Gauge32	Represents a fixed point value with six decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 123.456789

(Sheet 2 of 6)

Table 11 (continued)
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
	FixedPoint7	Gauge32	Represents a fixed point value with seven decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 12.3456789
	FixedPoint8	Gauge32	Represents a fixed point value with eight decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 1.23456789
	FixedPoint9	Gauge32	Represents a fixed point value with nine decimal places. For example, the value 123456789 should be displayed, and otherwise treated, as the decimal value: 0.123456789
HEX	Hex	Gauge32	Represents a decimal value that should be displayed in hexadecimal format.
INT_SEQUENCE		OBJECT IDENTIFIER,	Represents a variable length attribute of type INT_SEQUENCE and the value represents a real object identifier.
	IntegerSequence	OCTET STRING	Represents a fixed length attribute of type INT_SEQUENCE and the value does not map to a real object identifier.
IP_ADDRESS		IpAddress	Represents an octet string of 4 where the ip address numbers ranging from 0 to 255 is encoded in network order. For example, 47.102.38.38, octet[1] =47, octet[2]=102, octet[3]=38, octet[4]=38.
(Sheet 3 of 6)			

Table 11 (continued)
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
LINK	Link	OBJECT IDENTIFIER	Link is used to signal that the object identifier is a component and that the component has a relationship to the present component. (See “Links” (page 110).)
LONG_INTEGER	PassportCounter64	Counter64	SMIv1 only supports 32 bit integer values. Low order 32 bit integer returned.
SIGNED		INTEGER	Represents a signed integer.
STRING (ASCII)	AsciiString	OCTET STRING (SIZE (0..1024))	Represents a string of printable Ascii characters. One character encoded in Ascii per octet.
STRING (DASHED_HEX)	DashedHexString	OCTET STRING (SIZE (0..1024))	Represents a string of HEX digits. Two HEX digits are encoded per octet. The string should be formatted with dashes in between octets for display.
STRING (EXTENDED_ ASCII)	ExtendedAsciiString	OCTET STRING (SIZE (0..1024))	Represents an OCTET STRING that should be displayed as ASCII where there is a correspondence to an ASCII value and as `\xx` otherwise.
STRING (HEX)	HexString	OCTET STRING (SIZE (0..1024))	Represents string of HEX digits encoded two per OCTET.
(Sheet 4 of 6)			

Table 11 (continued)
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
TIME	EnterpriseDateAndTime	OCTET STRING (SIZE (0 | 2 | 5 | 8 | 10 | 13 | 16 | 19))	A date-time specification. Octets Contents Range ========== 1-4 year 0000..9999 5 '-' na 6-7 month 01..12 8 '-' na 9-10 day 01..31 11 ' ' (space) na 12-13 hour 00..23 14 ':' na 15-16 minutes 00..59 17 ':' na 18-19 seconds 00..60 Only those fields applicable will be present. An invocation of this textual convention will include the size limitation appropriate for the corresponding time precision. The correspondence between the length of the OCTET STRING and the time precision is: 2 - corresponds to H (Octets 12-13 present) 5 - corresponds to HM (Octets 12-16 present) 8 - corresponds to HMS (Octets 12-19 present) 10 - corresponds to D (Octets 1-10 present) 13 - corresponds to DH (Octets 1-13 present) 16 - corresponds to DHM (Octets 1-16 present) 19 - corresponds to DHMS (Octets 1-19

(Sheet 5 of 6)

Table 11 (continued)
Mapping of Passport CDL data types to Passport Enterprise MIB data types

Passport CDL data type	Passport SNMP textual convention	Base SNMP type	Description
WILD BCD	WildcardedDigitString	OCTET STRING (SIZE (0..1024))	Represents a string of decimal digits, that is 0..9 or the question mark character '?' that acts as a wildcard character for any single digit. Each OCTET is encoded with the ASCII value of a single digit or the ASCII value of the wild card character, '?'.
(Sheet 6 of 6)			

The table, "Mapping Passport component types to Passport Enterprise MIB types" (page 106), provides a listing of the mappings.

Table 12
Mapping Passport component types to Passport Enterprise MIB types

Passport CDL Data Type	Passport SNMP Textual Convention	Base SNMP Type	Display Hint	Example
Ascii String	AsciiString	OCTET STRING (SIZE (0..1024))	String of printable Ascii characters in the range 32 to 127.	For octet string, 66 72 6F 67 67 79, display froggy.
Bcd String	DigitString	OCTET STRING (SIZE (0..1024))	String of decimal digits (0..9). For each octet display ascii representation of digit.	For octet string, 31 32 33 34 35, display 12345.
Component Name	RowPointer	OBJECT IDENTIFIER	A valid object identifier that maps to an object in the MIB.	
	Compound Multi-Index			
(Sheet 1 of 5)				

Table 12 (continued)
Mapping Passport component types to Passport Enterprise MIB types

Passport CDL Data Type	Passport SNMP Textual Convention	Base SNMP Type	Display Hint	Example
Dashed Hex String	Dashed Hex String	OCTET STRING (SIZE (0..1024))	For each octet display in hex with dash (-) separators.	For octet string, 31 32 3D 4F, display 31-32-3D-4F.
Enum		INTEGER	Map integer value back to enumerated type. Display corresponding enum value.	Given a mapping between a set of names and integer values are: froggy(1), kerope(2) and grenouille(3). Then If integer value is 2, display kerope.
Extended Ascii String	ExtendedAsciiString	OCTET STRING (SIZE (0..1024))	For each octet display as for an AsciiString and /xx for octets that are not within the range of printable Ascii characters (32 to 127).	For octet string, 66 72 6F 67 67 79 16, display froggy/xx.
Fixed Point	FixedPoint1	Unsigned32	fixed point value with one decimal places	123456789 displayed as 12345678.9
	FixedPoint2	Unsigned32	fixed point value with two decimal places	123456789 displayed as 1234567.89
	FixedPoint3	Unsigned32	fixed point value with three decimal places	123456789 displayed as 123456.789
	FixedPoint4	Unsigned32	fixed point value with four decimal places	123456789 displayed as 12345.6789

(Sheet 2 of 5)

Table 12 (continued)
Mapping Passport component types to Passport Enterprise MIB types

Passport CDL Data Type	Passport SNMP Textual Convention	Base SNMP Type	Display Hint	Example
	FixedPoint5	Unsigned32	fixed point value with five decimal places	123456789 displayed as 1234.56789
	FixedPoint6	Unsigned32	fixed point value with six decimal places	123456789 displayed as 123.456789
	FixedPoint7	Unsigned32	fixed point value with seven decimal places	123456789 displayed as 12.3456789
	FixedPoint8	Unsigned32	fixed point value with eight decimal places	123456789 displayed as 1.23456789
	FixedPoint9	Unsigned32	fixed point value with nine decimal places	123456789 displayed as 0.123456789
Hex String	HexString	OCTET STRING (SIZE (0..1024))	For each octet display in hex.	For octet string, 31 32 3D 4F, display 31323D4F.
Hex Unsigned	Hex	Unsigned32	Display in hexadecimal format.	For 32 display as hex value 00 00 00 20.
Integer Sequence		OBJECT IDENTIFIER,		
	Integer Sequence	OCTET STRING	Each octet is a digit or a dot (.) separator.	For octet string 49 46 49 53 54 46 53 46 54 51 display as 1.156.5.63
Ip Address	IpAddress	OCTET STRING (SIZE (4))		47.208.132.114
(Sheet 3 of 5)				

Table 12 (continued)
Mapping Passport component types to Passport Enterprise MIB types

Passport CDL Data Type	Passport SNMP Textual Convention	Base SNMP Type	Display Hint	Example
Link	Link	OBJECT IDENTIFIER	A valid object identifier that maps to an object in the MIB.	1.3.6.1.4.1.562
Long Integer		OBJECT IDENTIFIER		
	Integer Sequence	OCTET STRING		
Set of Bits		OCTET STRING (SIZE (0..1024))	For each octet display as Hex or bits 0 and 1's.	
Signed Integer		INTEGER	Display as an integer.	
Time	EnterpriseDate AndTime	OCTET STRING (SIZE (0 | 2 | 5 | 8 | 10 | 13 | 16 | 19))	See Passport Textual Conventions for encoding and display hint.	
Unsigned Integer		Gauge32	Display as an Unsigned32. Gauge assigned to values which fluctuate between a lower and upper bound.	

(Sheet 4 of 5)

Table 12 (continued)
Mapping Passport component types to Passport Enterprise MIB types

Passport CDL Data Type	Passport SNMP Textual Convention	Base SNMP Type	Display Hint	Example
		Counter32	Display as an Unsigned32. Counter assigned to values which montonically increase.	
Wild Bcd String	WildcardedDigit String	OCTET STRING (SIZE (0..1024))	String of decimal digits (0..9) or the question mark (?). For each octet display ascii representation of a digit or ?.	For octet string, 31 32 33 34 3F 3F, display 1234??.
(Sheet 5 of 5)				

Links

The containment relationship cannot be used to model all the different kinds of relationships between components. Links represent relationships between components in different component hierarchies. These relationships are typically of the service-user/service-provider type, such as the link from a *Framer* to a V.35 port (refer to the figure “Component mapping example” (page 80)). Passport links are always bidirectional.

Variables representing links are of type *Link.* Values of these variables are always component names.

Links are used to

- support redundancy as logical processors run on physical cards and a card may have a back-up. Modeling the relationship between the logical processors and the cards as links simplifies the management of all aspects of redundant operations.

- represent common information since the logical processor type (LPT) defines the software running on a logical processor. Modeling this relationship as a link simplifies the provisioning of a new *Lp* component, or modifying the configuration of existing ones.
- represent a protocol stack, and in particular to bind a service onto some particular hardware. This simplifies hardware sparing, supports independent surveillance of service and hardware hierarchies and facilitates provisioning by enabling copy operations.

Component instance data types

Each component can be replicated by a single or a compound index. The component instance is made up of values of SNMP variables defined as indices. The following list indicates all the Passport CDL types supported as index values:

- BCD
- DECIMAL
- Enumerated Types
- HEX
- INT_SEQUENCE
- IP_ADDRESS
- STRING (ASCII)
- STRING (DASHED_HEX)
- STRING (HEX)
- WILD_BCD

See the table, “Mapping of Passport CDL data types to Passport Enterprise MIB data types” (page 101), for mapping of Passport CDL data types to Passport Enterprise MIB data types.

Mapping an index value to an instance object identifier

SNMP index values are specified by the management station as object identifiers. The table, “Mapping of SNMP index value into an object identifier” (page 112), describes how index values are mapped to object identifiers.

Table 13
Mapping of SNMP index value into an object identifier

Passport CDL Index Data Type	SNMP Instance Object Identifier
BCD	Specify the length of the bcd string, followed by the decimal ascii value of each digit in the bcd string.
DECIMAL	Use value as is
Enumerated Types	Use the decimal representing the enumerated value.
HEX	Convert the hex value to a decimal.
INT_SEQUENCE	Specify the length of the integer sequence followed by the value of each integer in the sequence.
IP_ADDRESS	Use value as is.
Non replicated components	Use 1
STRING (ASCII)	Specify the length of the ascii string followed by the decimal ascii value of each character. For characters that are alphabet letters, use upper case ascii values only. Receding and trailing blanks are ignored.
STRING (DASHED_HEX)	Specify the length of the hex string followed by the decimal value of each octet. Each octet is encoded with two hex values.
STRING (HEX)	Specify the length of the hex string followed by the decimal value of each octet. Each octet is encoded with two hex values.
WILD_BCD	Specify the length of the wild bcd string, followed by the decimal ascii value of each digit or '?' in the wild bcd string.

For compound indices, the SNMP instance object identifier is the concatenation of all the object identifiers mapped from each SNMP index variable value.

Special MIB modules

There are a number of special MIB modules, which provide global information relevant to the PASSPORT Passport Enterprise MIB.

- *Nortel-MsCarrier-MscPassport-UsefulDefinitionsMIB* defines the upper reaches of the Nortel Carrier Passport MIB structure (see “SNMP object identifier usage” (page 221)) and Passport defined values for *sysObjectID*.
- *Nortel-MsCarrier-MscPassport-StandardTextualConventionsMIB* provides a specification of all supported standard textual conventions. This is provided so Passport Enterprise MIB users do not have to search for, acquire, and compile the source Standard MIBs themselves.
- *Nortel-MsCarrier-MscPassport-TextualConventionsMIB* is the Textual Conventions specific to the Passport Enterprise MIB (see “Textual conventions” (page 229)).
- *Nortel-MsCarrier-MscPassport-AlarmMIB* provides a specification of Passport alarms.
- *Nortel-MsCarrier-MscPassport-ExtensionsMIB* provides extensions to certain Standard MIBs. For example, the MIB II *ifTable* has been extended.

Chapter 5
Passport Enterprise MIB management

This chapter contains a general description of the management of Passport Enterprise MIBs, including

MIB loading

Since the IMPORTS statements in the MIBs, and the MIB compilers or linkers, are of the one-pass variety, the order when loading MIBs into the SNMP managers is important. The loading order for Enterprise MIBs is as follows

1 nortelPC-usefulDefinitions<Vn>_CA02A.mib

2 nortelPC-standardTextualConventions<Vn>_CA02A.mib

3 nortelPC-textualConventions<Vn>_CA02A.mib

4 nortelPC-alarm<Vn>_CA02A.mib

5 nortelPC-extensions<Vn>_CA02A.mib

6 nortelPC-ruvStateSumm<Vn>_CA02A.mib

where:
<Vn> is the current SNMP version loaded on a Passport node. The current version can only be V1 or V2.

Since the dependency list for the remaining Enterprise MIBs varies between releases, please refer to the README file on the software distribution site (SDS) for a full list of these dependencies.

In some cases, Passport Enterprise MIB modules are organized into hierarchies analogous to the component hierarchy. This normally happens when the MIB modules are associated with related features that can be independently loaded.

The figure, "MIB module hierarchies" (page 117), illustrates the following:

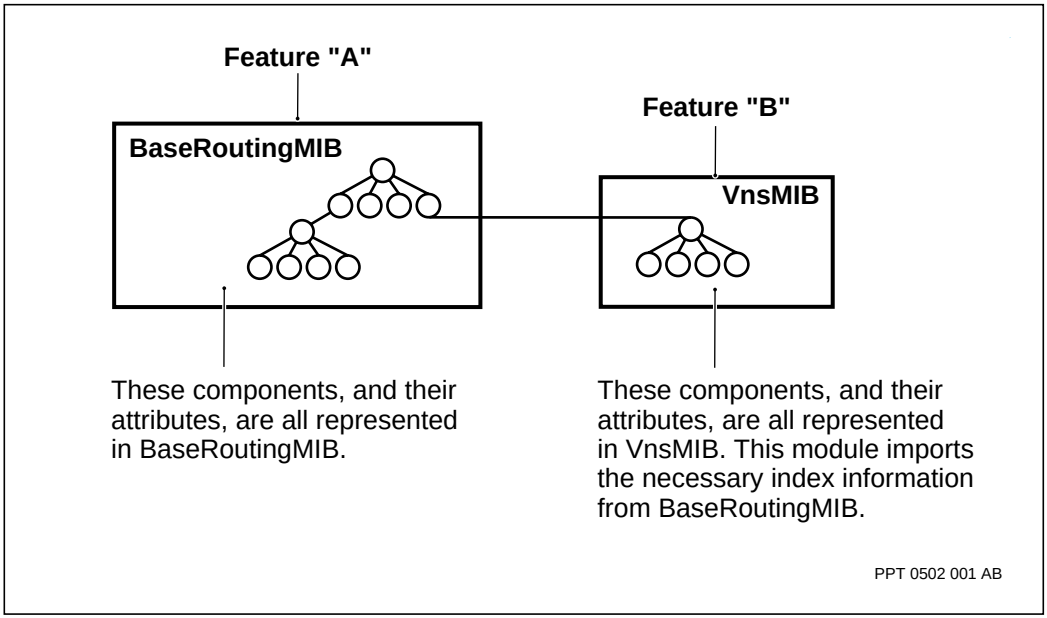
- *BaseRoutingMIB*, associated with Feature A, contains the definitions of the higher level components of the component hierarchy. This module is called the superior module.
- *VnsMIB*, associated with Feature B, contains the definitions of some components that are subordinate, in the component hierarchy, to components defined in *VnsMIB*. This module is called the subordinate module.

Subordinate modules import definitions from superior modules; these subordinate modules cannot be compiled alone. For example, the module *VnsMIB* is subordinate to the module *BaseRoutingMIB*. The *VnsMIB* contains the following imports clause:

```
mscRtgIndex, mscRtg
FROM Nortel-MsCarrier-MscPassport-BaseRoutingMIB
```

Note: Superior modules must be loaded before subordinate modules. For more information, see the description of the README file in "MIB loading" (page 115).

Figure 25
MIB module hierarchies



MIB distribution

Two different kinds of MIB files are involved in the distribution of Passport Enterprise MIBs.

- Passport Enterprise MIB files in text file format, for compiling and loading of SNMP manager applications.
- Passport Enterprise MIB files in on-node software files format for loading on Passport.

The figure, “Passport Enterprise MIB distribution” (page 119), illustrates MIB distribution. MIB distribution is organized as follows:

1 All Passport software, including Passport Enterprise MIBs in text format and on-node software, is delivered on CD-ROM. The software and MIBs are installed, from CD-ROM, onto the SDS workstation.

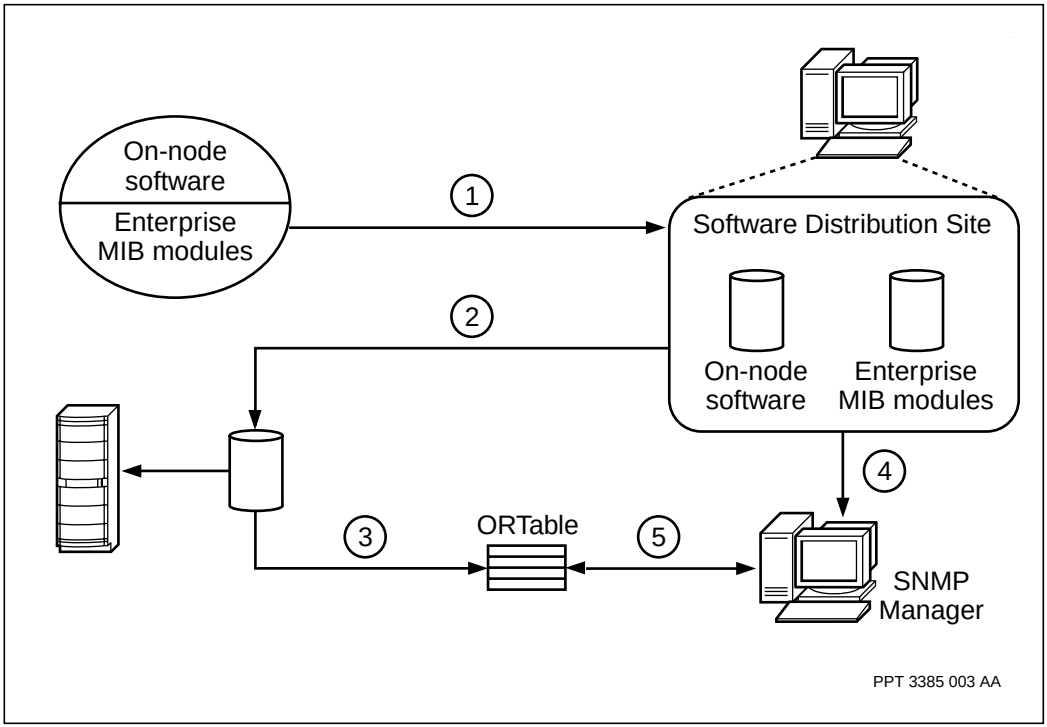
2 Passport Enterprise MIBs in on-node software are downloaded to a node as part of the download of associated application software. This happens automatically; no additional operator intervention is required with respect to MIB software.

3 On-node MIB software is loaded when the application and feature with which it is associated is provisioned (that is, when an LPT is provisioned with that feature). For example, if the Networking application is downloaded to a node, and the DPRS feature is then provisioned for an LPT supporting that application, the software for the *DpnRoutingMIB* module is also loaded. This updates the *ORTable* to reflect the presence of these modules, and the MIB variables become available to the manager, through the agent, for that node.

4 The Enterprise MIBs in text format associated with a feature must be compiled and loaded separately onto the SNMP manager. The procedures for this vary from manager to manager.

5 Managers can determine the Passport Enterprise MIBs loaded on a node, including their versions, by reading the *Object Resources Group table (ORTable)* defined in RFC 1907 or by displaying the provisioned applications. This is termed MIB discovery and is discussed in “MIB discovery” (page 121).

Figure 26
Passport Enterprise MIB distribution



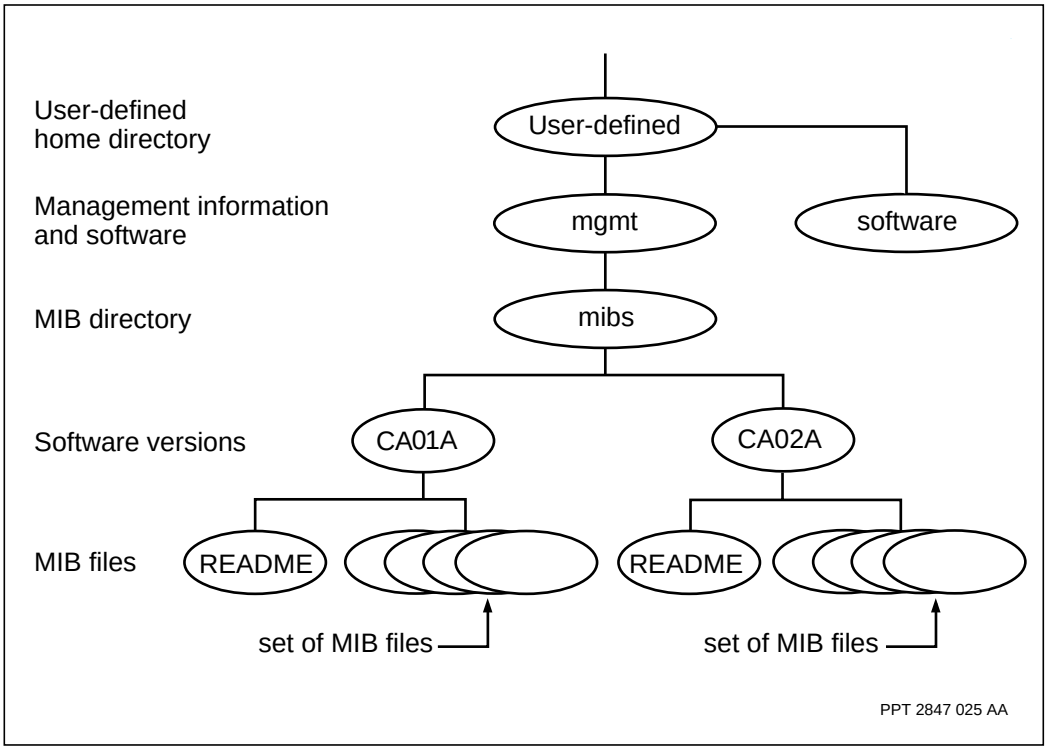
MIB files on the software distribution site

All SNMP related MIB files are available from the SDS. These files are available on a per software version basis. See the figure “Passport 7400, 15000 Enterprise MIB files at a software distribution site” (page 120). Each software version directory contains all the Enterprise MIBs for all applications released at that software version, and a README file. The README file provides

- an index of all Enterprise MIBs, organized by application
- the order in which to load the Enterprise MIBs into a SNMP manager

Files containing MIBs are named based on the name of the MIB module and the version of the software. For example, the directory *.../mgmt/mibs/V1/CA02A* contains the DPN routing feature. Please refer to the figure "MIB module file name convention" (page 121).

Figure 27
Passport 7400, 15000 Enterprise MIB files at a software distribution site



You should realize the difference between the version of a MIB (described in "MIB discovery" (page 121)) and the version of the software or application. The software distribution site's MIBs or software version directory always contains the complete set of Enterprise MIBs (regardless of whether the MIB version has or has not changed). For example, the following two files may be exactly the same MIB version.

mibs/<Vn>/BE01A/nortelPC-dpnRouting<Vn>_CA01.mib

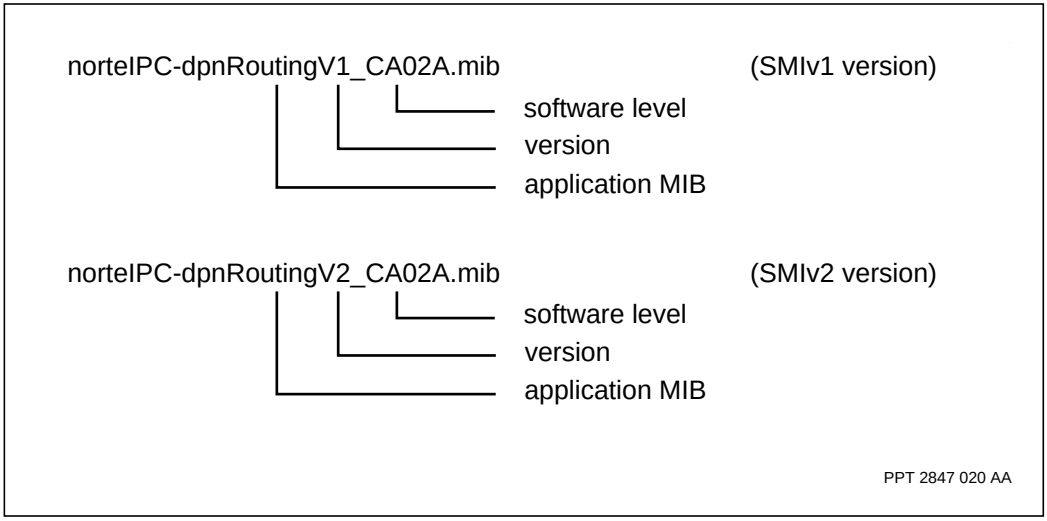
and

mibs/<Vn>/BF00A//nortelPC-dpnRouting<Vn>_CA02A.mib

where:
<Vn> is the current SNMP version loaded on a Passport node. The current version can only be V1 or V2.

Only editorial changes are allowed without changing the MIB version. To determine the version of MIB, refer to “MIB discovery” (page 121).

Figure 28
MIB module file name convention



MIB discovery

A manager must know the schema of the MIB modules supported by an agent running on a particular node in order to properly manage that node. This applies to every node for which the manager is responsible. MIB discovery is the term used here to describe the mechanisms provided to the manager by the agent to support discovery of its schema.

Note: MIB discovery refers to the discovery of MIB *schema*, and the simple term discovery refers to discovery of nodes and/or instance-values from the MIBs.

MIB discovery using SNMP

MIB discovery is supported through the SNMP "Object Resources Group" defined in RFC 1907. This group defines a collection of variables allowing a SNMPv2 entity acting in an agent role to describe the MIB modules it has loaded.

A manager can assess the available information on a particular node by

- examining the value of the *sysObjectID* variable to determine the product, and hence potential associated Standard and Enterprise MIBs. See "The Passport msCarrier object identifier tree" (page 222) for more information.
- examining the values of the *ifType* variable to determine the applicable Standard MIBs for the interfaces and their underlying media
- examining the values of the *ORTable* to determine current configured capabilities

These mechanisms are *standard* mechanisms.

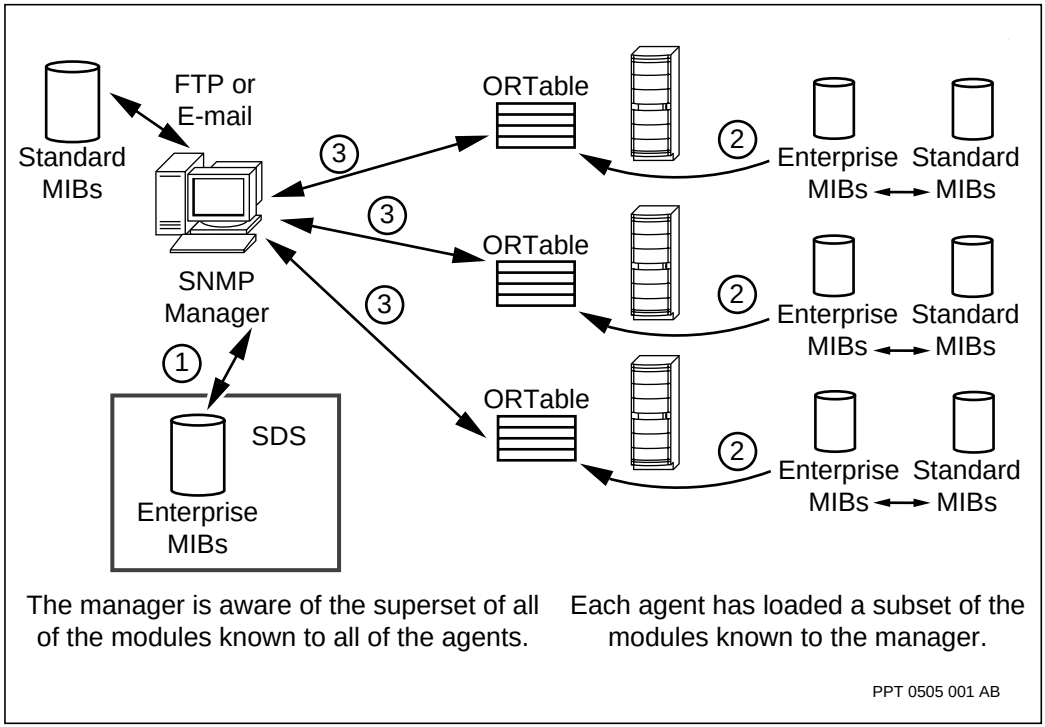
Only the *ORTable* allows for the precise communication of the schema of supported MIBs, a requirement for configuration management. This facility may be used for MIB discovery as illustrated in the figure "MIB discovery" (page 123):

1 Every Enterprise MIB module has an AGENT-CAPABILITIES statement. This statement is assigned an OBJECT IDENTIFIER value which identifies not only the MIB module but also the implementation or MIB version and Standard MIB modules support as well. Managers obtain these from the ASCII MIB modules available on the SDS.

2 When a Passport feature is loaded on a node, the MIB modules associated with the feature are also loaded. In conjunction with this, the *ORTable* is updated to reflect the fact that these MIB modules are present.

3 A manager may read the *ORTable* from any node. The manager may periodically poll the variable *sysORLastChange* variable to determine the last time the *ORTable* has changed.

Figure 29
MIB discovery



After loading RFC 1907, you can discover and retrieve MIBs using both SNMP and a telnet text interface.

To discover and retrieve MIBs using SNMP:

1 Sweep the *snmSysORTable* to determine the AGENT-CAPABILITIES of all V2 MIB modules loaded on the Passport node.

```
manager> walk snmSyspORID snmpSysORDescr

snmpSysORID.1       .1.3.6.1.4.1.562.36.2.2.4.3.1
snmpSysORDescr.1    "alarmCapabilitiesCA02A : mgmt/
mibs/CA02A/nortelPC-alarmV1_CA02A.mib"

snmpSysORID.2       .1.3.6.1.4.1.562.36.2.2.5.3.1
snmpSysORDescr.2    "extensionsCapabilitiesCA02A :
mgmt/mibs/CA02A/nortelPC-extensionsV1_CA02A.mib"
```

```
snmpSysORID.3      .1.3.6.1.4.1.562.36.2.2.2.3.1
snmpSysORDescr.3
"textualConventionsCapabilitiesCA02A : mgmt/mibs/
CA02A/nortelPC-textualConventionsV1_CA02A.mib"

...
```

Note: The *snmpSysORDescr* is a string containing the name and version of the AGENT-CAPABILITY associated with each Enterprise MIB module, followed by the path name, on the SDS site, of the file containing this MIB module. For example, <feature>Capabilities<mibVersion> would be mgmt/mibs/<softwareVersion><feature>V1_<softwareVersion>.mib.

2 Retrieve all the MIB files referred to in the *snmpORDescr* values from the SDS.

```
% ftp <sdsSite>
Connected to <sdsSite>
Name: <userid>
Password: <password>
ftp> get mgmt/mibs/CA02A/nortelPC-alarmV1_CA02A.mib
alarmV1_CA02A.mib
ftp> get mgmt/mibs/CA02A/nortelPC-
extensionsV1_CA02A.mib extensionsV1_CA02A.mib
ftp> quit
%
```

To discover and retrieve MIBs using a telnet text interface:

1 Display the software version of the applications running on the Passport node.

```
display -p sw

Sw
avList = base_CA02A networking_CA02A frameRelay_CA02A
ok
```

2 Retrieve all MIB files for the identified software version from the SDS.

```
% ftp <sdsSite>
Connected to <sdsSite>
Name: <userid>
Password: <password>
ftp> cd mgmt/mibs/CA02A
ftp> get README
ftp> prompt off
```

```
ftp> mget *
ftp> quit
%
```

MIB evolution

Passport MIB modules evolve in strict adherence to the rules of the SNMP SMI with respect to changing the OBJECT IDENTIFIER assignments made to variables. These rules, and the Passport rules for changing the OBJECT IDENTIFIER assignments made to AGENT-CAPABILITIES statements, are described in the following two sections.

Agent-capabilities statements

The AGENT-CAPABILITIES statement is a standard mechanism used to describe the capabilities provided by an agent.

Unchanged agent-capabilities

A variable definition may be revised in any of the following ways without changing either its reference name or its OBJECT IDENTIFIER value, and without causing the AGENT-CAPABILITIES or OBJECT-GROUP statements to change:

- A DEFVAL clause may be added.
- Editorial changes to any DESCRIPTION clause may be made.

Changed agent-capabilities

An object definition may be revised in any of the following ways without changing either its reference name or its OBJECT IDENTIFIER value. However, the AGENT-CAPABILITIES statement and OBJECT-GROUP definition do change:

- A SYNTAX clause containing an enumerated INTEGER may have new enumerations added or existing labels changed.
- A STATUS clause value of current may be revised as deprecated or obsolete. Similarly, a STATUS clause value of deprecated may be revised as obsolete.
- A conceptual row may be augmented by adding new columnar objects at the end of the row.

- Entirely new variables may be defined, named with previously unassigned OBJECT IDENTIFIER values.

Any other change to a variable definition requires a change to the OBJECT IDENTIFIER value assigned to that definition, and to the OBJECT IDENTIFIER value assigned to the AGENT-CAPABILITIES statement for that module.

If the semantics of any variable are changed (that is, if a non-editorial change is made other than those specifically allowed above), then the OBJECT IDENTIFIER value associated with that variable is changed.

Changing the reference name associated with an existing variable is considered a semantic change, as these strings may be used in an IMPORTS statement.

See the figure “Example agent-capabilities and object-group versioning” (page 127).

MIB versioning

The Passport SNMP Enterprise MIB versions are changed when there have been non-editorial changes. See “Changed agent-capabilities” (page 125).

The versioning of AGENT-CAPABILITIES statements and OBJECT-GROUP definitions is depicted in terms of the Passport MIB structure. Only leaf arcs indicated in bold are assigned to AGENT-CAPABILITIES statements. Assignments to OBJECT-GROUP definitions work exactly the same, except the string Group is used instead of Capabilities.

As shown in the figure “Example agent-capabilities and object-group versioning” (page 127), four different kinds of version changes are allowed:

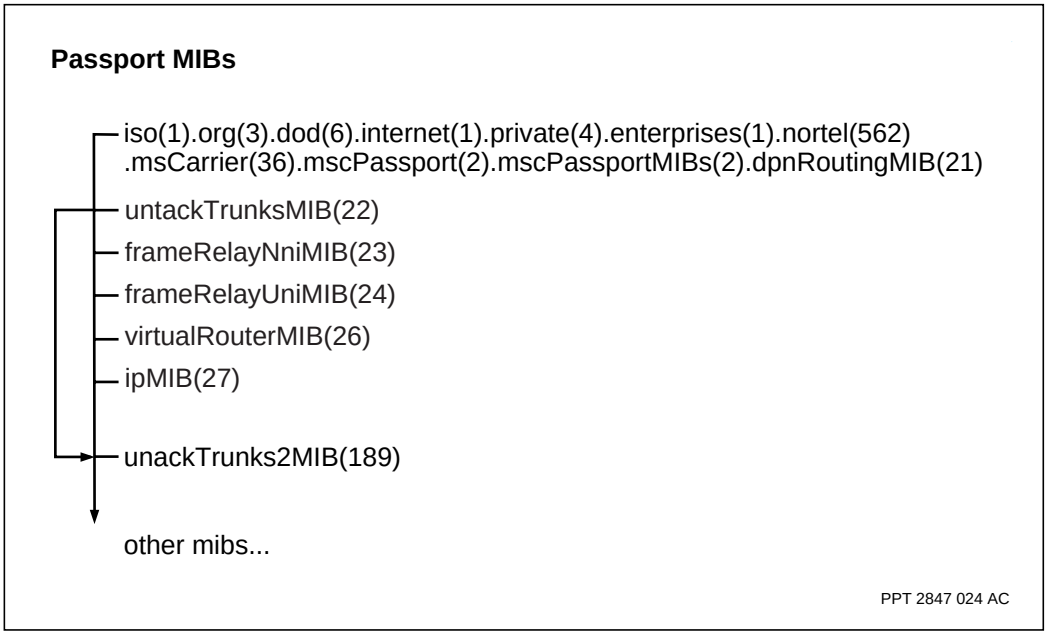
1 as a result of an SSUP

2 as the result of a RSUP

3 as the result of a new Release

4 a complete MIB re-assignment as the result of major changes to a MIB module. In this case, the MIB module is assigned a new arc and a new name.

Figure 30
Example agent-capabilities and object-group versioning



The values assigned to the AGENT-CAPABILITIES and OBJECT-GROUP statements are arranged in lexicographical order for a particular MIB module. This allows a manager to determine whether a particular OBJECT-IDENTIFIER represents an older or newer version of a known MIB, or if it represent a completely unknown MIB.

Arc numbers are assigned based on the following specific algorithm:

- At the release level the arc value is the same as the release.
- At the RSUP level the arc value is the RSUP level plus one. This is a special case as RSUP levels can be 00 and to avoid SNMP arc values of zero the RSUP level is always incremented by one.
- At the SSUP level the arc value starts at two and is incremented by one for each subsequent SSUP.

Variables that are no longer supported from older MIBs are kept in the MIB with a status of obsolete. The obsolete variables are removed from the MIB when a new MIB is issued.

This approach allows managers to:

- determine exactly what MIB modules are loaded onto a node. The manager has the ability to determine the exact MIB version running on a given node, via the mechanisms described in “MIB discovery” (page 121)
- plan for managing older MIBs, by using the *ORTable* to determine the correct version of the management application software to use with that node
- plan for managing newer schema, by using the *ORTable* to determine if a newer version of a known MIB module has been loaded on the node. In this scenario the manager must be prepared to encounter differences between the loaded and known MIB
- determine that new MIB modules must be downloaded from the SDS if unknown identifiers are discovered

Chapter 6
SNMP traps

Passport supports an alarm reporting facility for fault management of Passport nodes. Passport provides the management station with information on Passport alarms generated by mapping the alarms into an SNMP trap. The Passport Enterprise MIB-defined trap provides

- early detection of faults prior to damage.
- an indication of the level to which the fault degrades the expected quality of service.
- an alarm minimization philosophy. Alarms direct the manager to the direct cause of the problem, not the effects. This results in fewer alarms, each providing specific corrective actions.
- troubleshooting information. Alarms include information that helps in understanding the problem and providing potential repair actions.This information is available in a structured, machine-processable format modeled in the Enterprise MIB, and in an ASCII format.
- a generic format for the reporting of alarms. Six trap types are defined (one for each level of alarm severity) in the same format, providing:
  - an active list status. Alarms are cleared when the problem is resolved.
  - an index into the alarms NTP (241-5701-500 *Passport 6400, 7400, 15000, 20000 Alarms*)
  - other common parameters including the ASCII name of the component generating the alarm, the alarm type, severity, and probable cause

  - — in some cases, additional application- or component-specific machine-processable information particular to that specific alarm
- alignment with the OSI alarm reporting function, as defined in ITU-T Recommendation X.733 / ISO 10164-4.

Passport implements the SNMP standard interfaces group as defined by RFC 1213 and RFC 1573. This group provides generic information about the interfaces including configuration information, state, and statistics. Since not all interfaces are supported, see “IfTable” (page 143) for a list of the interfaces supported by Passport and detailed descriptions of how the interfaces group is supported by each supported interface.

This chapter discusses the following aspects of SNMP traps:

Standard traps

Some Standard MIBs define application-specific standard traps. The table, “Standard MIB traps” (page 131), lists the specific traps currently supported.

Table 14
Standard MIB traps

MIB name	RFC	Trap name	Description
A Simple Network Management Protocol	1157	ColdStart	A ColdStart trap indicates the SNMP agent has been initialized; for example, when it is first added, when the shelf reboots, or the CP restarts.
A Simple Network Management Protocol	1157	WarmStart	A WarmStart trap indicates that the SNMP agent has been unlocked after it was locked, or when the *virtualRouter* component has been unlocked after it was locked.
A Simple Network Management Protocol	1157	LinkDown	A LinkDown trap indicates that a communications link has gone down.
A Simple Network Management Protocol	1157	LinkUp	A LinkUp trap indicates that a communications link has gone up.
A Simple Network Management Protocol	1157	AuthenticationFailure	An AuthenticationFailure trap indicates that the SNMP agent has received a protocol message that was not authenticated. The *EnableAuthenTraps* attribute of the *Snmp* component is used to enable or disable the transmission of this trap. The authentication of a SNMP protocol message fails if: • a bad community string is used • the source address does not match the transportAddress of *manager* components for that *community* component • the type of protocol message is not allowed given the *accessMode* of the *community* or the privileges of the *manager* component (for example, a set PDU is received from a manager which only has get privileges).

(Sheet 1 of 2)

Table 14 (continued)
Standard MIB traps

MIB name	RFC	Trap name	Description
A Simple Network Management Protocol	1157	External Gateway Protocol (EGP) Neighbor Loss	This trap signifies that an EGP neighbor (for whom the sending protocol entity was an EGP peer) has been marked down and the peer relationship no longer exists.
FR DTE	1315	Fr DLCI Status Change	This trap indicates that the virtual circuit has changed state. It has either been created or invalidated, or has toggled between the active and inactive states.

(Sheet 2 of 2)

Considerations

The agent address in an SNMPv1 or SNMPv2 trap-PDU is populated as follows:

1 If the *Snmp* component's *ipStack* attribute has the value *ipiFrIpiVc*, then the agent address contains the *ipAddress* provisioned for the *IpiFr* or *IpiVc* component.

2 If the *Snmp* component's *ipStack* attribute has the value *vrIp*, then the agent address contains the *ipAddress* corresponding to the port the trap is sent out on as there could be multiple IP ports with multiple logical interfaces for each port.

Refer to "Configuring the Snmp component" (page 26) for more details on configuring and enabling traps.

Passport alarms as traps

Passport alarms are defined in the MIB module *Nortel-MsCarrier-MscPassport-AlarmMIB*. The MIB structure for this module is illustrated in the figure "MIB structure for alarms" (page 134).

Alarms contain OSI-compliant fields to indicate such things as the type, severity, and cause of the alarm. They contain Passport-defined value-added information, such as documentation (and categorization) indexing (NTP index), alarm status (*set*, *message*, *clear*) and fields of ASCII and hex text containing useful information.

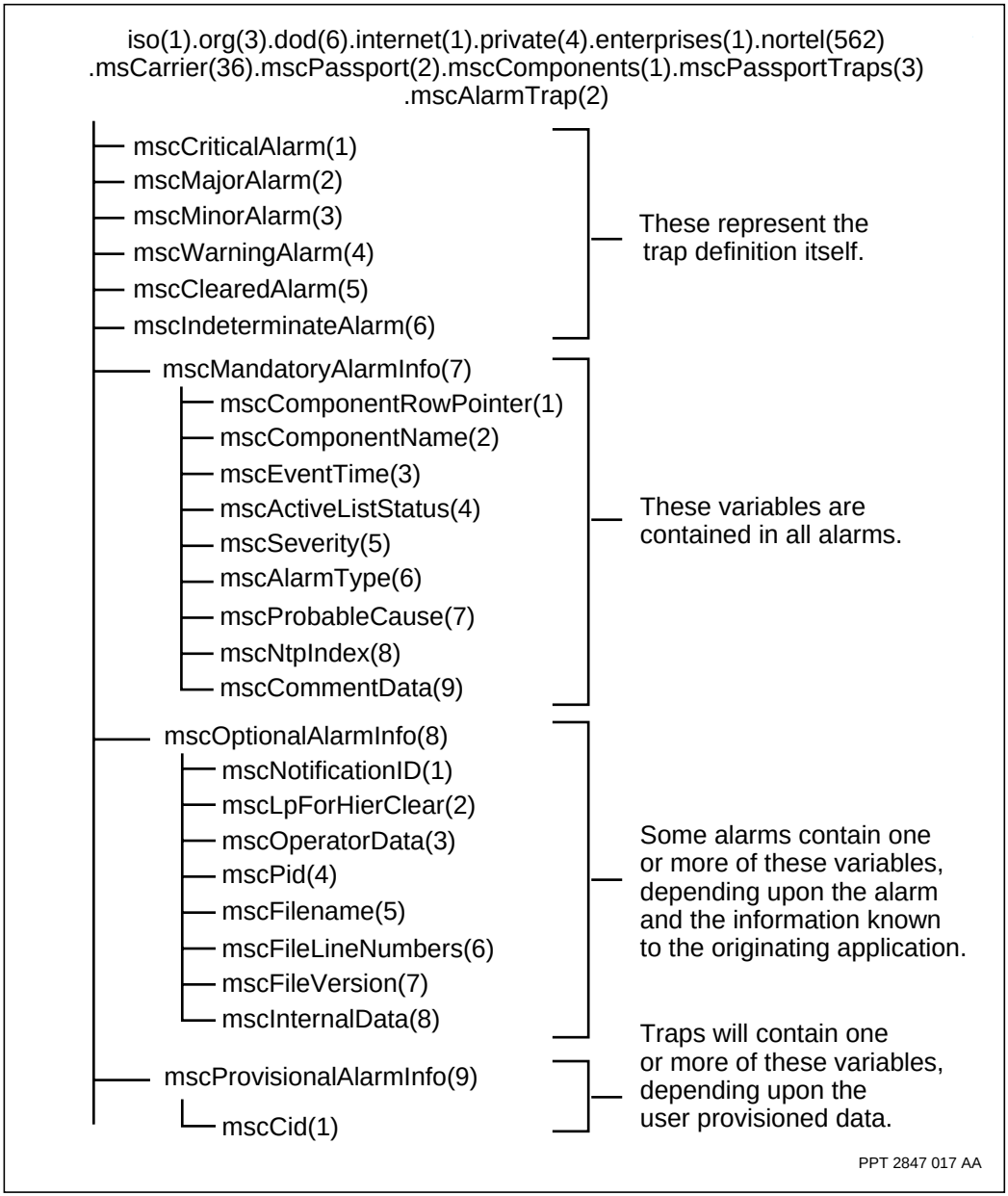
SNMP alarms do not contain any component state information.

In SNMP, alarms are transmitted as traps. Six traps are defined, each with the same structure and content. The traps are defined, one for each type of alarm severity:

- *mscCriticalAlarm*
- *mscMajorAlarm*
- *mscMinorAlarm*
- *mscWarningAlarm*
- *mscClearedAlarm*
- *mscIndeterminateAlarm*

These traps accommodate simple trap monitoring tools which assign severity to a trap based on the Specific Trap Number, rather than to the value of a variable bind in the trap. See “Standard traps” (page 130).

Figure 31
MIB structure for alarms



For example, a Passport alarm with a severity equal to critical is mapped into the *mscCriticalAlarm* notification.

> ***Note:*** The trap number corresponds to the severity variable enumerated value, with the exception of an *mscIndeterminateAlarm*, which has the trap number 6 and a severity variable value of 0.

Information on interpreting alarms and a detailed description of all possible Passport alarms are provided in 241-5701-500 *Passport 6400, 7400, 15000, 20000 Alarms*.

Trap definition

Passport alarms are defined as SNMPv1 or SNMPv2 traps using the TRAP-TYPE macro. See the table, “TRAP-TYPE macro information” (page 135), for specific information.

> ***Note:*** The variables listed in the VARIABLES clause, defined in the ‘mscMandatoryAlarmInfo’ group are present in all critical alarms. Additional variables as defined in the ‘mscOptionalAlarmInfo’ group may be included if appropriate. This applies to all attribute names listed in the following table.

Table 15
TRAP-TYPE macro information

Attribute name	Trap Number	Description
mscCriticalAlarm	1	This trap is used to provide a real time indication of a critical Alarm condition.
mscMajorAlarm	2	This trap is used to provide a real time indication of a major Alarm condition.
mscMinorAlarm	3	This trap is used to provide a real time indication of a minor Alarm condition.
mscWarningAlarm	4	This trap is used to provide a real time indication of a warning Alarm condition.

(Sheet 1 of 2)

Table 15 (continued)
TRAP-TYPE macro information

Attribute name	Trap Number	Description
mscClearedAlarm	5	This trap is used to provide a real time indication of a cleared Alarm.
mscIndeterminate Alarm	6	This trap is used to provide a real time indication of an indeterminate Alarm condition.

(Sheet 2 of 2)

Mandatory content

A trap always contains the following:

- *sysUptime* and *snmpTrapOID* as the first and second variables of the SNMPv2-Trap-PDU *varBindList*

  Note: The *sysUptime* and *snmpTrapOID* variables are not contained in the SNMPv1-Trap-PDU *varBindList*

- the variables listed in the OBJECTS clause of the trap definition. These are the same variables as defined in the *mscMandatoryAlarmInfo* group
- optionally, selected variables defined in the *mscOptionalAlarmInfo* group

The *mscMandatoryAlarmInfo* group, contained in all SNMP alarms, is defined as a SNMP info group. See the table, “mscMandatoryAlarmInfo group” (page 137), for specific details.

Table 16
mscMandatoryAlarmInfo group

Attribute name	Arc number	Description
mscComponentRow Pointer	1	A pointer to the RowStatus variable of the component emitting the alarm.
mscComponentName	2	The textual name of the component emitting the alarm in a format suitable for use in commands entered on the `console'. It includes the name of the node from which the alarm was sent, as specified in `ModuleData nodeName'.
mscEventTime	3	The date and time at which the alarmed event occurred.
mscActiveListStatus	4	The active alarm list status. This may be used by the Manager to add (on receiving a set) and delete (on receiving a clear) alarms from the alarm displays.
mscSeverity	5	The severity of the alarm which should indicate what priority that fixing this problem should be given. All values are OSI-defined; refer to ITU-T X.733/ISO 10164-4 for a more complete description.
mscAlarmType	6	The alarm type field as defined by OSI which is used to indicate a broad category of what is wrong. The first 6 values are OSI-defined; refer to ITU-T X.733/ISO 10164-4 (for the first 5) and ITU-T X.736/ISO 10164-4 (for the last) for a more complete description. The value 'operator' is used when an alarm is issued due to an operator command. The values 'debug' and 'unknown' are for compatibility with older switches and are used for debugging alarms and for those which do not fit any of the above, respectively.
mscProbableCause	7	The probable cause for the alarm which usually qualifies the Alarm Type field. Most values are OSI-defined; refer to ITU-T X.733 and X.736 (ISO 10164-4 and 10164-7) for a more complete description.

(Sheet 1 of 2)

Table 16 (continued)
mscMandatoryAlarmInfo group

Attribute name	Arc number	Description
mscNtpIndex	8	An 8-digit code which is an index into an NTP to provide more description of the alarm and possible actions to take. The first 4 digits indicate a 'group' of alarms (belonging to a particular subsystem or component, etc.). The last 4 digits are a subindex arbitrarily assigned by the application which is responsible for them.
mscCommentData	9	Additional textual information for the network operator to use. It contains only ASCII characters. Note: The total number of bytes in commentData, operatorData, and internalData may not exceed 750 bytes.

(Sheet 2 of 2)

Optional content

Some alarms contain variables which are optional. The table, "mscOptionalAlarmInfo group" (page 139), describes these optional variables.

Table 17
mscOptionalAlarmInfo group

Attribute name	Arc number	Description
mscNotificationID	1	The notification Id of the alarm. The high-order byte is the card number of the card from which this alarm originated. The low-order 24 bits contain a sequence number which increases each time an alarm is generated. Thus, this id provides a simple way of uniquely identifying an alarm as well as the order in which alarms are generated on a card. For cross-card ordering, one would have to look at the time stamp (see dateAndTime).
mscLpForHierClear	2	A pointer to the RowStatus variable of the Logical Processor the component generating the alarm is running on. A component generating an Alarm with `activeListStatus = set' can include this variable in the Alarm. If the identified Logical Processor generates a Hierarchical Clear Alarm, then it will clear the Set Alarm.
mscOperatorData	3	Additional hexadecimal information for the network operator to use. Note: The total number of bytes in commentData, operatorData, and internalData may not exceed 750 bytes.
mscPid	4	A string representation of the internal process id (Pid) of the process which generated the alarm (for internal use only). Note that it may or may not map to the componentName field.
mscFileName	5	An internal filename (for internal use only).
mscFileLineNumber	6	An internal line number in the file (see filename) referred to above (for internal use only).
mscFileVersion	7	A version indicator of the file (filename) referred to above (for internal use only).
mscInternalData	8	An additional internal hexadecimal information for internal use only. Note: The total number of bytes in comment, operator, and internal data may not exceed 750 bytes.

Provisionable content

The customer identifier (CID) contains the customer identifier of the component that generated the alarm. You have the option to provision whether or not the trap contains the customer identifier data. For information on how to provision the CID, see “Configuring the Snmp component” (page 26).

The table,“mscProvisionalAlarmInfo group” (page 140), contains a description of the provisional group. This group contains provisional variables representing information pertinent to all alarms: CID.

Table 18
mscProvisionalAlarmInfo group

Attribute name	Arc number	Description
mscCid	1	This is the Customer Identifier(CID) of the component which generated the alarm.

Considerations

The related component names field, as a list of component names, does not map readily into a variable binding for a *trap*. It is not included in SNMP alarms. The main purpose for this field is to specify the name of an LP for hierarchical clearing of alarms, and so an *mscLpForHierClear* variable, with more precise semantics, is substituted. Managers must themselves be able to associate components generating alarms with the LP that they are on.

Hierarchical clears

A hierarchical clear clears all alarms in a subtree of the MIB. This clear condition is indicated when the *mscNtpIndex* has the value ‘0000 0000’. The root of the subtree is indicated by the *mscComponentRowPointer*. Only select components issue hierarchical clears, such as *Lp* clearing outstanding alarms when recovering from a failure.

In some cases a set alarm may provide an *mscLpForHierClear* value. In this case, a hierarchical clear on the indicated *Lp* also clears the *set* alarm containing the *mscLpForHierClear* value. This is useful for hierarchically clearing a set alarm for a component outside the *logicalProcessor* MIB subtree.

Chapter 7
IfTable

The interfaces group consists of the variable *ifNumber*, the *ifTable*, and the *ifXTable*. As this group is the most widely implemented SNMP group, it is the most widely used for troubleshooting multi-vendor networks.

The *ifTable* provides a basic model of a network device containing a set of interfaces. Currently, the *ifTable* and a proposal for its evolution are defined in two RFCs:

- *RFC 1213 Management Information Base for Network Management of TCP/IP-based internets: MIB-II*
- *RFC 1573 Evolution of the Interfaces Group of MIB-II*

The *ifXTable* extends the current Passport *ifTable* implementation, and provides support for 64-counters. The *ifXTable* specifications are defined in the following RFCs:

- *RFC 1573 Evolution of the Interfaces Group of MIB-II*
- *RFC 2863 The Interfaces Group MIB*

This chapter discusses the following aspects of the SNMP *ifTable* and *ifXTable*:

- "Variable definitions" (page 144)
- "Passport ifEntries" (page 149)
- "Considerations" (page 181)

For more information about configuration considerations and SNMP support of specific FPs, refer to 241-5701-615 *Passport 7400, 15000, 20000 FP Configuration Reference*.

Variable definitions

The interfaces group consists of the variable *ifNumber*, the *ifTable*, and the *ifXTable*.

The *ifNumber* variable represents the number of network interfaces (regardless of their current state) present on the system.

The *ifTable* maintains information for each of the interfaces on the system. Each interface is viewed as if it is attached to a subnetwork. The original definition of the interface (as defined in RFC 1213) states that an interface corresponds one-to-one with a physical port. RFC 1573 relaxes this definition, and describes an interface as any layer of a protocol stack. The Passport system's interpretation of an interface follows the definition in RFC 1573.

> ***Note:*** Buses, disks, file systems, and applications performing routing functions are not implemented in the *ifTable*. Only software entities that can be considered to be part of a protocol stack are implemented.

The table, "Variables of the ifTable from RFC 1213" (page 145), provides the attribute name, type, value, and description for the variables of each interface. The data for this table is obtained through a set of procedures; see "Uses of the ifTable" (page 202).

Table 19
Variables of the ifTable from RFC 1213

Attribute name	Type	Values	Description
ifIndex	Dec	0 to 65535	Represents a unique value for each interface. Other tables are indexed by variables mapping onto ifIndex, such as, atIfIndex (from the address translation group), ipNetToMediaIfIndex (from the IP group), various tables in the E1/DS1 and E3/DS3 MIBs, and various tables in the X.25 MIBs.
ifType	Enumeration	other, ddnX25, rfc877x25, ethernetCsmacd, ethernet8023, tokenBus8024, man802, starLan, hyperChannel, lapb, sdlc, ds1, e1, basicIsdn, primaryIsdn, propPointTo, PointSerial, ppp, ds3, smds, frameRelay	Represents the type of interface (as defined by MIB II).
ifAdminStatus	Enumeration	up, down, testing	A read-write attribute that represents the desired state of the interface.
ifOperStatus	Enumeration	up, down, testing	Represents the current operational state of the interface.
ifSpeed	Dec	0 to 4294967295	Represents an estimate of the interface's current bandwidth.
ifMtu	Dec	0 to 4294967295	Represents the interface's maximum transmission unit.
ifPhysAddress	ASCII string	0 to 17 ASCII characters	Represents the physical address (for example MAC address) of an interface.

(Sheet 1 of 3)

Table 19 (continued)
Variables of the ifTable from RFC 1213

Attribute name	Type	Values	Description
ifDescr	ASCII string	0 to 255 ASCII characters	Represents a textual string containing information about the interface. “Nortel: <ifTypeName> [<serial number> or <software version>]” where: ifTypeName is for example, trunk, dpnGate, DS1, or E3; serial number is used for ports and channels; and software version is used for all other services.
ifLastChange	Dec	0 to 4294967295	Represents the value of sysUpTime at the time the interface entered its current operational state.
ifSpecific	Integer Seq	Set to { 0 0 }. Deprecated in RFC 1573.	Represents a reference to MIB definitions specific to the particular media used to realize the interface.
ifInOctets	Dec	0 to 4294967295	Represents the total number of octets received on the interface, including framing characters.
ifInUcastPkts	Dec	0 to 4294967295	Represents the number of subnetwork-unicast packets delivered to a higher-layer protocol.
ifInNUcastPkts	Dec	0 to 4294967295	Represents the number of non-unicast packets (that is, subnetwork-broadcast or subnetwork-multicast) delivered to a higher-layer protocol.
ifInDiscards	Dec	0 to 4294967295	Represents the number of inbound packets that were discarded even though there were no errors.
ifInErrors	Dec	0 to 4294967295	Represents the number of inbound packets that could not be delivered to a higher-level protocol because of errors.
ifInUnknownProtos	Dec	0 to 4294967295	Represents the number of packets received via the interface that were discarded because of an unknown or unsupported protocol.

(Sheet 2 of 3)

Table 19 (continued)
Variables of the ifTable from RFC 1213

Attribute name	Type	Values	Description
ifOutOctets	Dec	0 to 4294967295	Represents the total number of octets transmitted out of the interface, including framing characters.
ifOutUcastPkts	Dec	0 to 4294967295	Represents the total number of packets that were transmitted (at the request of a higher-level protocol) to a subnetwork- unicast address, including those packets that were discarded or not sent.
ifOutNUcastPkts	Dec	0 to 4294967295	Represents the total number of packets that were transmitted (at the request of a higher-level protocol) to a non-unicast address (that is, a subnetwork-broadcast or subnetwork-multicast address). This total also includes the packets that were discarded or not sent.
ifOutDiscards	Dec	0 to 4294967295	Represents the number of outbound packets that were discarded even though no errors were detected.
ifOutErrors	Dec	0 to 4294967295	Represents the number of outbound packets that were not transmitted because of errors.
ifOutQLen	Dec	0 to 4294967295	Represents the aggregated queue length of the output packet queues on that interface.
ifName			From RFC 1573

(Sheet 3 of 3)

The *ifXTable*, defined in RFC 1573, augments the columns of the *ifTable*, and provides extended management information related to each interface. The *ifXTable* is indexed by *ifIndex* from the *ifTable*. The table “64-bit counters of the ifXTable from RFC 1573” (page 148) provides a summary of the additional objects that are available. These objects are 64-bit versions of the corresponding *ifTable* counters, with identical semantics.

Table 20
64-bit counters of the ifXTable from RFC 1573

Attribute name	Type	Values	Description
ifHCInOctets	Dec	0 to 18446744073709551615	Represents the total number of octets received on the interface, including framing characters. This object is a 64-bit version of ifInOctets.
ifHCOutOctets	Dec	0 to 18446744073709551615	Represents the total number of octets transmitted out of the interface, including framing characters. This object is a 64-bit version of ifOutOctets.
ifHCInUcastPkts	Dec	0 to 18446744073709551615	Represents the number of unicast packets (that is, subnetwork-broadcast or subnetwork-multicast) delivered to a higher-layer protocol. This object is a 64-bit version of ifInUcastPkts.
ifHCOutUcastPkts	Dec	0 to 18446744073709551615	Represents the total number of packets that were transmitted (at the request of a higher-level protocol) to a subnetwork- unicast address, including those packets that were discarded or not sent. This object is a 64-bit version of ifOutUcastPkts.

ifName of RFC 1573

The *ifName* variable, defined in RFC 1573, represents the textual name of the interface; its value is the name of the interface as assigned by the local device and can be used in commands entered at the device's console (for example, *le0*). In the Passport system, the *ifName* variable contains the component name associated with the *ifEntry*.

ifRowPointer of the Passport Enterprise MIB (Extensions MIB)

The Passport system supports a suite of Enterprise MIBs. Therefore, an additional mapping is required to go from a row of the *ifTable* to the corresponding row(s) of table(s) defined in the appropriate Passport Enterprise MIB. The *ifRowPointer* variable represents the *RowPointer* for the component corresponding to an ifIndex.

Passport ifEntries

The layer3 and layer2 interfaces implemented in the ifTable are summarized in the table “Layer 3 interfaces” (page 149) and in the table “Layer 2 interfaces” (page 150).

Table 21
Layer 3 interfaces

Layer 3 interface	Related information
x25Dte	“ifEntry for x25Dte and x25DteRemoteGroup components” (page 153)
x25DteRemoteGroup	“ifEntry for x25Dte and x25DteRemoteGroup components” (page 153)
FrDte	“ifEntry for FrDte and FrDteRemoteGroup components” (page 154)
FrDteRemoteGroup	“ifEntry for FrDte and FrDteRemoteGroup components” (page 154)
FrDce	“ifEntry for FrDce components” (page 155)
Frame Relay UNI	“ifEntry for LanApplication component” (page 158)
Frame Relay NNI	“ifEntry for LanApplication component” (page 158)
LanApplication	“ifEntry for LanApplication component” (page 158)

(Sheet 1 of 2)

Table 21 (continued)
Layer 3 interfaces

Layer 3 interface	Related information
PointToPointProtocol	"ifEntry for PointToPointProtocol component" (page 159)
Vs	"ifEntry for Vs, Btds, and Htds components" (page 160)
Btds	"ifEntry for Vs, Btds, and Htds components" (page 160)
Htds	"ifEntry for Vs, Btds, and Htds components" (page 160)
Trunk	"ifEntry for Trunk and DpnGate components" (page 161)
DpnGate	"ifEntry for Trunk and DpnGate components" (page 161)
Frame Relay to ATM interworking	"ifEntry for frame relay to ATM interworking components" (page 162)
MPANL	"ifEntry for MPANL component" (page 165)
FrMux	"ifEntry for FrMux component" (page 168)
(Sheet 2 of 2)	

Table 22
Layer 2 interfaces

Layer 2 interface	Related information
Lp Ethernet	"ifEntry for Lp OamEthernet and Lp Ethernet components" (page 171)
Lp OamEthernet	"ifEntry for Lp OamEthernet and Lp Ethernet components" (page 171)
(Sheet 1 of 3)	

Table 22 (continued)
Layer 2 interfaces

Layer 2 interface	Related information
Lp DS1 Lp DS3 DS1 Lp Sonet Sts Vt1dot5 DS1	"ifEntry for Lp DS1, Lp E1, Lp DS3 DS1, Lp E3 E1, Lp Sonet Sts Vt1dot5 DS1, and Lp Sdh Vc4 Vc12 E1 components" (page 173)
Lp E1 Lp E3 E1 Lp Sdh Vc4 Vc12 E1	"ifEntry for Lp DS1, Lp E1, Lp DS3 DS1, Lp E3 E1, Lp Sonet Sts Vt1dot5 DS1, and Lp Sdh Vc4 Vc12 E1 components" (page 173)
Lp DS1 Chan Lp DS3 DS1Chan Lp Sonet Sts Vt1dot5 DS1 Chan Lp E1 Chan Lp E3 E1 Chan Lp Sdh Vc4 Vc12 E1 Chan	"ifEntry for Lp DS1 Chan, Lp E1 Chan, Lp DS3 DS1 Chan, Lp E3 E1 Chan, Lp Sonet Sts Vt1dot5 DS1 Chan, Lp Sdh Vc4 Vc12 E1 Chan component" (page 174)
Lp DS3	"ifEntry for Lp DS3 and Lp E3 components" (page 176)
Lp E3	"ifEntry for Lp DS3 and Lp E3 components" (page 176)
Lp V35	"ifEntry for Lp V35 and Lp X21 components" (page 178)
Lp X21	"ifEntry for Lp V35 and Lp X21 components" (page 178)
Lp Sonet	"ifEntry for Lp Sonet and Lp Sdh components" (page 179)
Lp Sdh	"ifEntry for Lp Sonet and Lp Sdh components" (page 179)
Lp Sonet Path	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)

(Sheet 2 of 3)

Table 22 (continued)
Layer 2 interfaces

Layer 2 interface	Related information
Lp Sonet Sts	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)
Lp Sdh Vc4	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)
Laps Sts	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)
Lp Sdh Vc4 Vc12	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)
Lp Sonet Sts Vt1dot5	"ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components" (page 180)
(Sheet 3 of 3)	

The semantics of all variables do not apply to all interfaces. The following tables list the level of support for each variable for the Passport interfaces.

Note: Unless otherwise defined, the significance of each variable is as defined in the table "Variables of the ifTable from RFC 1213" (page 145).

Table 23
ifEntry for x25Dte and x25DteRemoteGroup components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: X25 DTE Interface software version” or “Nortel: X25 Dte Remote Group software version”
ifType	Y	The value for X25 DTE is “rfc877_x25(5)” The value for X25 DTE Remote Group is “Other=1”
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	
ifAdminStatus	Y	up, down, testing
ifOperStatus	Y	up, down, testing
ifLastChange	Y	
ifInOctets	N	
ifInUcastPkts	N	
ifInNUcastPkts	N	
ifInDiscards	N	
ifInErrors	N	
ifInUnknownProtos	N	
ifOutOctets	N	
ifOutUcastPkts	N	
ifOutNUcastPkts	N	
ifOutDiscards	N	
ifOutErrors	N	
ifOutQLen	N	
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

Table 24
ifEntry for FrDte and FrDteRemoteGroup components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: Frame Relay DTE software version” or “Nortel: FrDteRemote Group software version”
ifType	Y	The type of interface. The value for Frame Relay DTE is “frameRelay(32)” The value for FrDteRemote Group is “Other=1”
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	
ifAdminStatus	Y	up, down, testing
ifOperStatus	Y	up, down, testing
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	
ifInNUcastPkts	Y	
ifInDiscards	Y	
ifInErrors	Y	
ifInUnknownProtos	N	
ifOutOctets	Y	
ifOutUcastPkts	Y	
ifOutNUcastPkts	Y	
ifOutDiscards	Y	
ifOutErrors	Y	
ifOutQLen	N	
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

Table 25
ifEntry for FrDce components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: Frame Relay Service: software version”
ifType	Y	The type of interface. The value for Frame Relay DCE is “frameRelayService(44)”
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface. It is set to the Maximum Frame Length in the Framer associated with the logical port.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	The interface's address at the protocol layer immediately `below' the network layer. Set to zero length String
ifAdminStatus	Y	The desired (provisioned) state of the FR DCE logical port. Value is up, down, testing
ifOperStatus	Y	The current operational state of the FR DCE logical port. Value is up, down, testing.
ifLastChange	Y	The elapsed time since the last re-initialization of the logical port. The value of sys Up Time at the time the FR DCE logical port entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value.
ifInOctets	Y	The total number of octets received on the FR DCE logical port. This counter counts octets from the beginning of the frame relay header, user data, and the 2 octets for CRC check.
ifInUcastPkts	Y	The number of received non-errored unicast frames.
ifInNUcastPkts	N	The number of received non-errored multicast frames. Always set to zero.
ifInDiscards	Y	The number of received frames discarded. Possible reasons are: policing, congestion. Sum of all received discarded frames over all DLCIs at this logical port (e.g. rate enforcement, rate adaptation, congestion, A-bit not available).

(Sheet 1 of 2)

Table 25 (continued)
ifEntry for FrDce components

Variable	Support	Comments/Values
ifInErrors	Y	The number of received frames discarded because of an error. Possible errors can be the following: frame relay frames are too long or too short, frames had an invalid or unrecognized DLCI value, or incorrect header values. Sum of all discarded frames at the FR DCE logical port (for example, due to unknown DLCI) and over all DLCIs at this logical port (e.g. frames too long or too short).
ifInUnknownProtos	N	The number of frames received via the interface which were discarded because of an unknown or unsupported protocol. FR DCE logical ports will always set this value to zero.
ifOutOctets	Y	The total number of transmitted octets on the FR DCE logical port. This counter counts octets from the beginning of the frame relay header, user data, and the 2 octets for CRC check.
ifOutUcastPkts	Y	The number of frames sent.
ifOutNUcastPkts	N	The number of sent non-errored multicast frames. Always set to zero.
ifOutDiscards	Y	The number of frames discarded in the transmit direction. Possible reasons are: policing, congestion. Aggregation of all transmitted discarded frames over all DLCIs at this logical port (for example, transmit discards due to congestion).
ifOutErrors	Y	The number of frames discarded in the transmit direction because of errors. Always set to zero.
ifOutQLen	N	The length of the output frame queue (in packets). Always set to zero.
ifSpecific	N	A reference to MIB definition particular to media used. Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	The component instance name, for example, FrUni/x or FrNni/x

(Sheet 2 of 2)

Table 26
ifEntry for Frame Relay UNI and NNI components

Variable	Support	Comments/Values
ifDescr	Y	“NORTEL - Frame Relay X:Y”, where X is either UNI or NNI, and Y is the software version number.
ifType	Y	The value is 44.
ifMtu	Y	Set to MaximumFrameLength set in the Framer associated with the FR logical port.
ifSpeed	Y	Set to the actual speed of the underlying interface or channel.
ifPhysAddress	N	
ifAdminStatus	Y	up, down, testing
ifOperStatus	Y	up, down, testing
ifLastChange	Y	Set to zero.
ifInOctets	Y	Includes the two-byte DLCI header and the two-byte CRC.
ifInUcastPkts	Y	
ifInNUcastPkts	N	Always set to zero.
ifInDiscards	Y	Represents the sum of all ingress discarded frames over all DLCIs at this logical port.
ifInErrors	Y	Represents the sum of all discarded frames at the frame relay logical port.
ifInUnknownProtos	N	Always set to zero.
ifOutOctets	Y	Includes the two-byte DLCI header and the two-byte CRC.
ifOutUcastPkts	Y	
ifOutNUcastPkts	N	
ifOutDiscards	Y	Aggregation of all egress discarded frames over all DLCIs at this logical port.
ifOutErrors	N	Always set to zero; no frames are discarded due to errors.

(Sheet 1 of 2)

Table 26 (continued)
ifEntry for Frame Relay UNI and NNI components

Variable	Support	Comments/Values
ifOutQLen	N	Set to zero.
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

(Sheet 2 of 2)

Table 27
ifEntry for LanApplication component

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: LanApplication software version”
ifType	Y	The value is “Other = 1”
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	
ifAdminStatus	Y	up, down, testing
ifOperStatus	Y	up, down, testing
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	
ifInNUcastPkts	N	
ifInDiscards	Y	
ifInErrors	Y	
ifInUnknownProtos	Y	
ifOutOctets	Y	
ifOutUcastPkts	Y	

(Sheet 1 of 2)

Table 27 (continued)
ifEntry for LanApplication component

Variable	Support	Comments/Values
ifOutNUcastPkts	Y	
ifOutDiscards	Y	
ifOutErrors	N	
ifOutQLen	N	
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

(Sheet 2 of 2)

Table 28
ifEntry for PointToPointProtocol component

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: PointToPointProtocol software version”
ifType	Y	The value is “PPP(23)”
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	
ifAdminStatus	Y	up, down, testing
ifOperStatus	Y	up, down, testing
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	
ifInNUcastPkts	Y	
ifInDiscards	Y	
ifInErrors	Y	

(Sheet 1 of 2)

Table 28 (continued)
ifEntry for PointToPointProtocol component

Variable	Support	Comments/Values
ifInUnknownProtos	N	
ifOutOctets	Y	
ifOutUcastPkts	Y	
ifOutNUcastPkts	Y	
ifOutDiscards	Y	
ifOutErrors	Y	
ifOutQLen	N	
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

(Sheet 2 of 2)

Table 29
ifEntry for Vs, Btds, and Htds components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: Btds software version”, “Nortel: Htds software version”, or “Nortel: Vs software version”
ifType	Y	The value for all three services is “Other = 1”.
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	Set to zero length String.
ifAdminStatus	Y	Value is up, down, or testing.
ifOperStatus	Y	Value is up, down, or testing.
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	

(Sheet 1 of 2)

Table 29 (continued)
ifEntry for Vs, Btds, and Htds components

Variable	Support	Comments/Values
ifInNUcastPkts	N	Set to 0.
ifInDiscards	Y	Represents the number of inbound packets that were discarded even though there were no errors.
ifInErrors	N	Set to 0.
ifInUnknownProtos	N	Set to 0.
ifOutOctets	Y	
ifOutUcastPkts	Y	
ifOutNUcastPkts	N	Set to 0.
ifOutDiscards	Y	Represents the number of outbound packets that were discarded even though no errors were detected.
ifOutErrors	N	Set to 0.
ifOutQLen	N	Set to 0.
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

(Sheet 2 of 2)

Table 30
ifEntry for Trunk and DpnGate components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: Trunk software version” or “Nortel: DpnGate software version”.
ifType	Y	The value for both services is “Other” (1).
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	Set to zero length String.

(Sheet 1 of 2)

Table 30 (continued)
ifEntry for Trunk and DpnGate components

Variable	Support	Comments/Values
ifAdminStatus	Y	Value is up, down, or testing.
ifOperStatus	Y	Value is up, down, or testing.
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	
ifInNUcastPkts	N	Unknown, set to 0.
ifInDiscards	Y	
ifInErrors	N	Inaccessible to Trunk and DpnGate. Set to 0.
ifInUnknownProtos	N	Unknown, set to 0.
ifOutOctets	N	Only known by UTP and DpnGate. Set to 0.
ifOutUcastPkts	N	Unknown, set to 0.
ifOutNUcastPkts	N	Unknown, set to 0.
ifOutDiscards	N	Only UTP knows of discards due to congestion. Set to 0.
ifOutErrors	N	Unknown, set to 0.
ifOutQLen	N	Unknown, set to 0.
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

(Sheet 2 of 2)

Table 31
ifEntry for frame relay to ATM interworking components

Variable	Support	Comments/Values
IfDescr	Y	“Nortel: Frame Relay Service: software version”
ifType	Y	The type of interface. IfType for FR-ATM Service is “frameRelayService(44).

(Sheet 1 of 3)

Table 31 (continued)
ifEntry for frame relay to ATM interworking components

Variable	Support	Comments/Values
ifMtu	Y	Represents the size of the largest datagram that can be sent or received on the interface. It is set to Maximum Frame Length in the Framer associated with the FR-ATM IWF logical port.
ifSpeed	Y	Represents an estimate of the interface's current bandwidth in bits per second. Set to the actual speed of the underlying interface or channel.
ifPhysAddress	N	The interface's address at the protocol layer immediately 'below' the network layer. Set to zero length string.
ifAdminStatus	Y	The desired state of the FR-ATM IWF logical port. Value is up, down, testing.
ifOperStatus	Y	The current operational state of the FR-ATM IWF logical port. Value is up, down, testing.
ifLastChange	Y	The elapsed time since the last re-initialization of the logical port. The value of sys Up Time at the time the FR-ATM IWF logical port entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value.
ifInOctets	Y	The total number of octets received on the FR-ATM logical port. This counter counts octets from the beginning of the frame relay header, user data, and the 2 octets for CRC check.
ifInUcastPkts	Y	The number of received non-errored unicast frames.
ifInNUcastPkts	N	The number of received non-errored multicast frames. Always set to zero.
ifInDiscards	Y	The number of received frames discarded. Possible reasons are: policing, congestion. Sum of all received discarded frames over all DLCIs at this logical port (for example, rate enforcement, rate adaptation, congestion, A-bit not available).

(Sheet 2 of 3)

Table 31 (continued)
ifEntry for frame relay to ATM interworking components

Variable	Support	Comments/Values
ifInErrors	Y	The number of received frames discarded because of an error. Possible errors can be the following: frame relay frames are too long or too short, frames had an invalid or unrecognized DLCI values, or incorrect header values. Sum of all discarded frames at the FR-ATM IWF logical port (for example, due to unknown DLCI) and over all DLCIs at this logical port (for example, frames too long or too short).
ifInUnknownProtos	N	The number of frames received via the interface which were discarded because of an unknown or unsupported protocol. FR-ATM logical ports will always set this value to zero.
ifOutOctets	Y	The number of transmitted octets. This counter counts octets from the beginning of the frame relay header, user data, and the 2 octets for CRC check.
ifOutUcastPkts	Y	The number of frames sent.
ifOutNUcastPkts	N	The number of sent non-errored multicast frames. Always set to zero.
ifOutDiscards	Y	The number of frames discarded in the transmit direction. Possible reasons are: policing, congestion. Aggregation of all transmitted discarded frames over all DLCIs at this logical port (for example, transmit discards due to congestion).
ifOutErrors	Y	The number of frames discarded in the transmit direction because of errors. Always set to zero.
ifOutQLen	N	The length of the output frame queue (in packets). Always set to zero.
ifSpecific	N	A reference to MIB definition particular to media used. Set to { 0 0 }.
ifName	Y	The component instance name; for example, FrAtm/x

(Sheet 3 of 3)

Table 32
ifEntry for MPANL component

Variable	Support	Comments/Values
IfDescr	Y	“Multiservice Passport Access Network Link:<sw version>”
ifType	Y	The type of interface. Value will be “Other = 1”
ifMtu	Y	The size of the largest datagram which can be sent or received on the interface. The size of the datagram sent on the interface can be different from the size of the datagram received by the interface. Tunneled or Dedicated or ISDN modes: the maxFrameSize signaled by the Passport 4400 or the default if one was not signaled.
ifSpeed	Y	An estimate of the interface’s current bandwidth in bits per second. Although an Mpanl interface transmit bandwidth is limited to the Transmit Information Rate (TIR), it’s receive bandwidth can be greater. Tunneled mode: the value of the TIR signaled by the Passport 4400 or the default if one was not signaled. Dedicated/ISDN mode: the actual line speed of the underlying interface or channel.
ifPhysAddress	Y	The interface’s address at the protocol layer immediately ‘below’ the network layer. Not applicable. Set to zero string length.
ifAdminStatus	Y	The desired state of the interface. up = interface is operational down = interface is not operational testing = no operational packets can be passed
ifOperStatus	Y	The current operational state of the interface. up = interface is operational down = interface is not operational testing = no operational packets can be passed

(Sheet 1 of 3)

Table 32 (continued)
ifEntry for MPANL component

Variable	Support	Comments/Values
ifLastChange	Y	The value of sys Up Time at the time the interface entered its current operational state.
ifInOctets	Y	The total number of octets received on the interface. This counter only counts octets from the beginning of the frame relay (MPANL) header (including the two-byte DLCI header) field to the end of the user data. This is the value reported by Vofr not Framer.
ifInUcastPkts	Y	The number of received unicast frames. This is the value reported by Vofr not Framer.
ifInNUcastPkts	Y	The number of non-unicast packets delivered to a higher layer protocol. Not applicable as this interface could never receive broadcast or multicast type packets. Always set to zero.
ifInDiscards	Y	The number of frames which were chosen to be discarded in the ingress direction even though no errors had been detected. Possible reasons are: congestion. Sum of all ingress discarded frames over all DLCIs at this interface.
ifInErrors	Y	The number of frames discarded in the ingress direction because of error. Possible errors include: frames were too long or too short, frames had an invalid or unrecognized DLCI value, frame contained incorrect header values, protocol violations of frames received detected by Vofr, or frames discarded due to lost fragments. Sum of all ingress discarded frames due to errors over all DLCIs at this interface plus due to errors reported by Vofr.
ifInUnknownProtos	Y	The number of packets received via the interface which were discarded because of an unknown or unsupported protocol. Not applicable. Always set to zero.

(Sheet 2 of 3)

Table 32 (continued)
ifEntry for MPANL component

Variable	Support	Comments/Values
ifOutOctets	Y	The total number of octets transmitted out of the interface. This counter only counts octets from the beginning of the frame relay header (including the two-byte DLCI header) field to the end of the user data. This is the value reported by Vofr not Framer.
ifOutUcastPkts	Y	The number of unicast frames sent on the interface. This is the value reported by Vofr not Framer.
ifOutNUcastPkts	Y	The total number of packets that higher level protocols requested be transmitted to a non-unicast (that is, a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent. Not applicable as this interface could never transmit broadcast or multicast type packets. Always set to zero.
ifOutDiscards	Y	The number of frames discarded in the egress direction. Possible reasons are: congestion. Sum of all egress discarded frames over all DLCIs at this interface.
ifOutErrors	Y	The number of outbound packets that could not be transmitted because of errors. Not applicable. Always set to zero.
ifOutQLen	Y	The length of the output packet queue (in packets). Not applicable. Always set to zero.
ifSpecific	Y	A reference to MIB definition particular to media used. Not applicable. Set to { 0 0 }.
ifName	Y	The textual name of the interface: “Mpanl/<instance number>”

(Sheet 3 of 3)

Table 33
ifEntry for FrMux component

Variable	Support	Comments/Values
IfDescr	Y	“Nortel-Frame Relay Muxer:<sw version>”
ifType	Y	The type of interface.
		IfType will be “FrameRelay = 32”
ifMtu	Y	The size of the largest datagram which can be sent/received on the interface.
		The maximumFrameLength set in the Framer associated with this interface.
ifSpeed	Y	An estimate of the interface’s current bandwidth in bits per second. Although FrMux’s interface transmit bandwidth is limited to the aggregate of all Committed Information Rates (CIR) of the tunneling DLCIs, it’s receive bandwidth can be larger.
		Set to the actual line speed of the underlying interface or channel.
ifPhysAddress	Y	The interface’s address at the protocol layer immediately ‘below’ the network layer.
		Not applicable. Set to zero string length.
ifAdminStatus	Y	The desired state of the interface.
		up = interface is operational
		down = interface is not operational
		testing = no operational packets can be passed
ifOperStatus	Y	The current operational state of the interface.
		up = interface is operational
		down = interface is not operational
		testing = no operational packets can be passed
ifLastChange	Y	The value of sys Up Time at the time the interface entered its current operational state.

(Sheet 1 of 3)

Table 33 (continued)
ifEntry for FrMux component

Variable	Support	Comments/Values
ifInOctets	Y	The total number of octets received on the interface. This counter only counts octets from the beginning of the frame relay header (including the two-byte DLCI header) field to the end of the user data.
ifInUcastPkts	Y	The number of received unicast frames.
ifInNUcastPkts	Y	The number of non-unicast packets delivered to a higher layer protocol. Not applicable as this interface could never receive broadcast or multicast type packets. Always set to zero.
ifInDiscards	Y	The number of frames which were chosen to be discarded in the ingress direction even though no errors had been detected. Possible reasons are: congestion. Sum of all ingress discarded frames over all DLCIs at this interface.
ifInErrors	Y	The number of frames discarded in the ingress direction because of error. Possible errors include: frames were too long or too short, frames had an invalid or unrecognized DLCI value, or frame contained incorrect header values. Sum of all ingress discarded frames due to errors over all DLCIs at this interface.
ifInUnknownProtos	Y	The number of packets received via the interface which were discarded because of an unknown or unsupported protocol. Not applicable. Always set to zero.
ifOutOctets	Y	The total number of octets transmitted out of the interface. This counter only counts octets from the beginning of the frame relay header (including the two-byte DLCI header) field to the end of the user data.
ifOutUcastPkts	Y	The number of unicast frames sent on the interface.

(Sheet 2 of 3)

Table 33 (continued)
ifEntry for FrMux component

Variable	Support	Comments/Values
ifOutNUcastPkts	Y	The total number of packets that higher level protocols requested be transmitted to a non-unicast (that is, a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent. Not applicable as this interface could never transmit broadcast or multicast type packets. Always set to zero.
ifOutDiscards	Y	The number of frames discarded in the egress direction. Not applicable since there is no queueing at the FrMux level. It is up to the application to queue and hence discard if congestion occurs. Always set to zero.
ifOutErrors	Y	The number of outbound packets that could not be transmitted because of errors. Not applicable. Always set to zero.
ifOutQLen	Y	The length of the output packet queue (in packets). Not applicable. Always set to zero.
ifSpecific	Y	A reference to MIB definition particular to media used. Not applicable. Set to { 0 0 }.
ifName	Y	The textual name of the interface: “FrMux/<instance number>”

(Sheet 3 of 3)

Table 34
ifEntry for Lp OamEthernet and Lp Ethernet components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifDescr	Y	Y	“Nortel: LAN Ethernet serial number” or “Nortel-CardType:CardType(serialNumber)”
ifType	Y	Y	The value for Lp Ethernet and LPOamEthernet is “ethernetCsmacd(6)”
ifMtu	Y	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	Y	N	
ifAdminStatus	Y	N	Value is up, down, or testing.
ifOperStatus	Y	Y	Value is up, down, or testing.
ifLastChange	N	N	
ifInOctets	Y	N	
ifInUcastPkts	Y	N	
ifInNUcastPkts	Y	N	
ifInDiscards	Y	N	
ifInErrors	Y	N	
ifInUnknownProtos	Y	N	
ifOutOctets	Y	N	
ifOutUcastPkts	Y	N	
ifOutNUcastPkts	Y	N	
ifHCInOctets	N	N	Defined in the table “64-bit counters of the ifXTable from RFC 1573” (page 148)

(Sheet 1 of 2)

Table 34 (continued)
ifEntry for Lp OamEthernet and Lp Ethernet components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifHCOutOctets	N	N	Defined in the table “64-bit counters of the ifXTable from RFC 1573” (page 148)
ifHCInUcastPkts	N	N	Defined in the table “64-bit counters of the ifXTable from RFC 1573” (page 148)
ifHCOutUcastPkts	N	N	Defined in the table “64-bit counters of the ifXTable from RFC 1573” (page 148)
ifOutDiscards	Y	N	
ifOutErrors	Y	N	
ifOutQLen	N	N	
ifSpecific	Y	N	MIB ID for Ethernet OamEthernet is 1.3.6.1.2.1.10.7
ifName	Y	Y	The component instance name; for example, Lp/0 OamEthernet/0

(Sheet 2 of 2)

Table 35
ifEntry for Lp DS1, Lp E1, Lp DS3 DS1, Lp E3 E1, Lp Sonet Sts Vt1dot5 DS1, and Lp Sdh Vc4 Vc12 E1 components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifDescr	Y	Y	“Nortel: DS1 software version” or “Nortel: E1 software version” or “Nortel-CardType:CardType(serialNumber) ”
ifType	Y	Y	DS1 or E1. IfType will be: “Ds1=18”, “E1=19”
ifMtu	Y	Y	Largest among all the MaxFrameLength for all the channel components under the port.
ifSpeed	Y	Y	Represents an estimate of the interface’s current bandwidth in bits per second. For DS1 = 1.544 Mbps, E1 = 2.048 Mbps.
ifPhysAddress	N	N	Set to zero length String.
ifAdminStatus	Y	N	Value is up, down, or testing.
ifOperStatus	Y	Y	Value is up, down, or testing.
ifLastChange	Y	N	
ifInOctets	Y	N	Aggregation of all channel ifInOctets defined under the port.
ifInUcastPkts	Y	N	Aggregation of all channel ifInUcastPkts defined under the port.
ifInNUcastPkts	N	N	Always set to 0.
ifInDiscards	N	N	Discards performed by hardware. Set to 0.
ifInErrors	Y	N	Aggregation of all channel ifInErrors defined under the port.
ifInUnknownProtos	N	N	Zero count
ifOutOctets	Y	N	Aggregation of all channel ifOutOctets defined under the port.

(Sheet 1 of 2)

Table 35 (continued)
ifEntry for Lp DS1, Lp E1, Lp DS3 DS1, Lp E3 E1, Lp Sonet Sts Vt1dot5 DS1, and Lp Sdh Vc4 Vc12 E1 components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifOutUcastPkts	Y	N	Aggregation of all channel ifOutUcastPkts defined under the port.
ifOutNUcastPkts	N	N	Always set to 0.
ifOutDiscards	N	N	Discards performed by hardware. Set to 0.
ifOutErrors	Y	N	Aggregation of all channel ifOutErrors defined under the port.
ifOutQLen	N	N	Set to 0.
ifSpecific	N	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	Y	The component instance name; for example, Lp/2 Ds1/0 or Lp/15 Sdh/0 Vc4/0 Vc12/1,1,1 E1

(Sheet 2 of 2)

Table 36
ifEntry for Lp DS1 Chan, Lp E1 Chan, Lp DS3 DS1 Chan, Lp E3 E1 Chan, Lp Sonet Sts Vt1dot5 DS1 Chan, Lp Sdh Vc4 Vc12 E1 Chan component

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifDescr	Y	Y	“Nortel: DS1 Chan serial number” or “Nortel: E1 Chan serial number” or “Nortel-CardType:CardType(serialNumber)”
ifType	Y	Y	Channel. IfType will be: “Channel =70”.

(Sheet 1 of 3)

Table 36 (continued)
ifEntry for Lp DS1 Chan, Lp E1 Chan, Lp DS3 DS1 Chan, Lp E3 E1 Chan, Lp Sonet Sts Vt1dot5 DS1 Chan, Lp Sdh Vc4 Vc12 E1 Chan component

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifMtu	Y	N	MaxFrameLength set in the Framer by the application. If there is no framer, it is set to 0 along with all other attributes collected from framer.
ifSpeed	Y	Y	Represents an estimate of the interface's current bandwidth in bits per second. It is derived from provisioned time slots.
ifPhysAddress	N	N	Set to zero string length.
ifAdminStatus	Y	N	Value is up, down, or testing.
ifOperStatus	Y	Y	Value is up, down, or testing.
ifLastChange	Y	N	
ifInOctets	Y	N	New data path subsystem provides the byte count for each frame.
ifInUcastPkts	Y	N	FrameFromIf derived from framer, total frames received from I/F.
ifInNUcastPkts	N	N	Always set to 0.
ifInDiscards	N	N	Discards performed by hardware. Set to 0.
ifInErrors	Y	N	CRC, overrun, nonOctets, abort, and Frames too long, errors derived from framer and aggregated.
ifInUnknownProtos	N	N	Zero count.
ifOutOctets	Y	N	New data path subsystem provides the byte count for each frame. It is set to 0 (not collected) if there is no outbound processing.
ifOutUcastPkts	Y	N	Frames TolF derived from framer, total frames transmitted to I/F.When there is no outbound processing, is set to 0.

(Sheet 2 of 3)

Table 36 (continued)
ifEntry for Lp DS1 Chan, Lp E1 Chan, Lp DS3 DS1 Chan, Lp E3 E1 Chan, Lp Sonet Sts Vt1dot5 DS1 Chan, Lp Sdh Vc4 Vc12 E1 Chan component

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifOutNUcastPkts	N	N	Always set to 0.
ifOutDiscards	N	N	Discards performed by hardware. Set to 0.
ifOutErrors	Y	N	LRC, underrun errors derived from framer and aggregated.
ifOutQLen	N	N	The block count in TxQueueSize in framer.
ifSpecific	N	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	Y	The component instance name; for example, Lp/2 Ds1/0Chan/0 or Lp/15 Sdh/0 Vc4/0 Vc12/1,1,1 E1 Chan/0.

(Sheet 3 of 3)

Table 37
ifEntry for Lp DS3 and Lp E3 components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifDescr	Y	Y	“Nortel: DS3 FP serial number” or “Nortel: E3 FP serial number” or “Nortel-CardType:CardType(serialNumber)”
ifType	Y	Y	DS3. IfType will be: “DS3=30”
ifMtu	Y	N	Represents the size of the largest datagram which can be sent or received on the interface.

(Sheet 1 of 2)

Table 37 (continued)
ifEntry for Lp DS3 and Lp E3 components

Variable	Support on Passport 7400	Support on Passport 15000 and 20000	Comments/Values
ifSpeed	Y	Y	Represents an estimate of the interface's current bandwidth; it is 44.736 Mbps.
ifPhysAddress	N	N	
ifAdminStatus	Y	N	Value is up, down, or testing.
ifOperStatus	Y	Y	Value is up, down, or testing.
ifLastChange	Y	N	
ifInOctets	Y	N	
ifInUcastPkts	Y	N	
ifInNUcastPkts	N	N	Always set to 0.
ifInDiscards	N	N	Discards performed by hardware. Set to 0.
ifInErrors	Y	N	
ifInUnknownProtos	N	N	Set to 0.
ifOutOctets	Y	N	
ifOutUcastPkts	Y	N	
ifOutNUcastPkts	N	N	Set to 0.
ifOutDiscards	N	N	Discards performed by hardware. Set to 0.
ifOutErrors	Y	N	
ifOutQLen	N	N	Set to 0
ifSpecific	N	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	Y	The component instance name; for example, Lp/2 DS3/0

(Sheet 2 of 2)

Table 38
ifEntry for Lp V35 and Lp X21 components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel: V35 FP serial number” or “Nortel: X21 FP serial number”
ifType	Y	The value is propPointToPointSerial(22) for both services.
ifMtu	Y	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth in bits per second.
ifPhysAddress	N	
ifAdminStatus	Y	Value is up, down, or testing.
ifOperStatus	Y	
ifLastChange	Y	
ifInOctets	Y	
ifInUcastPkts	Y	
ifInNUcastPkts	N	Always set to 0.
ifInDiscards	N	Discards performed by hardware. Set to 0.
ifInErrors	Y	
ifInUnknownProtos	N	Set to 0.
ifOutOctets	Y	
ifOutUcastPkts	Y	
ifOutNUcastPkts	N	Set to 0.
ifOutDiscards	N	Discards performed by hardware. Set to 0.
ifOutErrors	Y	
ifOutQLen	N	Set to 0
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	

Table 39
ifEntry for Lp Sonet and Lp Sdh components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel-CardType:CardType(serialNumber)”
ifType	Y	SonetSdh. IfType will be: “SonetSdh=39”
ifMtu	N	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidth.
ifPhysAddress	N	
ifAdminStatus	N	Value is up, down, or testing.
ifOperStatus	Y	Value is up, down, or testing.
ifLastChange	N	
ifInOctets	N	
ifInUcastPkts	N	
ifInNUcastPkts	N	Always set to 0.
ifInDiscards	N	Discards performed by hardware. Set to 0.
ifInErrors	N	
ifInUnknownProtos	N	Set to 0.
ifOutOctets	N	
ifOutUcastPkts	N	
ifOutNUcastPkts	N	Set to 0.
ifOutDiscards	N	Discards performed by hardware. Set to 0.
ifOutErrors	N	
ifOutQLen	N	Set to 0
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	The component instance name; for example, Lp/2 Sonet/0

Table 40
ifEntry for Lp Sonet Path, Lp Sonet Sts, Lp Sdh Vc4, Laps Sts, Lp Sdh Vc4 Vc12, and Lp Sonet Sts Vt1dot5 components

Variable	Support	Comments/Values
ifDescr	Y	“Nortel-CardType:CardType(serialNumber)”
ifType	Y	SonetPath. IfType will be: “SonetPath=50”
ifMtu	N	Represents the size of the largest datagram which can be sent or received on the interface.
ifSpeed	Y	Represents an estimate of the interface’s current bandwidths.
ifPhysAddress	N	
ifAdminStatus	N	Value is up, down, or testing.
ifOperStatus	Y	Value is up, down, or testing.
ifLastChange	N	
ifInOctets	N	
ifInUcastPkts	N	
ifInNUcastPkts	N	Always set to 0.
ifInDiscards	N	Discards performed by hardware. Set to 0.
ifInErrors	N	
ifInUnknownProtos	N	Set to 0.
ifOutOctets	N	
ifOutUcastPkts	N	
ifOutNUcastPkts	N	Set to 0.
ifOutDiscards	N	Discards performed by hardware. Set to 0.
ifOutErrors	N	
ifOutQLen	N	Set to 0
ifSpecific	N	Set to { 0 0 }. Deprecated in RFC 1573.
ifName	Y	The component instance name Lp/2 Sonet/0 Sts/0 or Lp/4 Sdh/0 Vc4/0

Considerations

You can relate *ifTable* information to a Passport component on the node through a telnet session by referring to the table "Displaying ifTable information" (page 181).

The CAS interface, accessed via telnet, does not use the prefix *msc* associated with Passport's SNMP object names in either the operator commands or in the resulting output displays. For example, the *VirtualRouter* component's object name is *mscVr* but is represented in the CAS interface as *Vr*. See "Mapping the component model to Passport Enterprise MIBs" (page 79) for more information.

Table 41
Displaying ifTable information

Type of information requested	Command to enter
Display information about the attributes of the ifTable using the help command.	h Vr IfTableEntry
Display provisionable values of a specific component..	d -p <name_of_component>/* ***Note:*** All components in the IfTable have an *ifIndex* attribute that can be received by displaying their provisionable attributes
Display the values of the IfEntires associated with one or all Vr components.	d Vr/* IfTableEntry/* (for multiple *Vr* components) d Vr/<n> IfTableEntry/* (for one *Vr* component) ***Note:*** You can also use this command to query 64-bit counter information from the *ifXTable*. The *ifXTable* information is concatenated at the end of the original *ifTableEntry* attributes.

(Sheet 1 of 2)

Table 41 (continued)
Displaying ifTable information

Type of information requested	Command to enter
Display the values of the ifAdminStatus variable for each Passport interface associated with a specified *Vr* component through the IfTable.	d Vr/<n> IfTableEntry/<n> ifAdminStatus
Display the values of the componentName variable for each Passport interface associated with a specified *Vr* component through the IfTable	d Vr/<n> IfTableEntry/<n> componentName

(Sheet 2 of 2)

Chapter 8
Surveillance

This chapter describes the different aspects of SNMP surveillance:

SNMP agent OSI state

The SNMP agent assumes various OSI states indicative of whether or not the agent can process traps and requests.

The *Snmp* component may assume the following operational states:

- enabled

  The IP UDP layer is up, and the virtual router under which the *Snmp* component resides is unlocked.

- disabled

  Either the IP UDP layer is down, or the virtual router under which the *Snmp* component resides is locked. On initial startup the *Snmp* component is disabled for 90 seconds to allow the IP UDP layer sufficient time to come up.

The *Snmp* component may assume the following administrative states:

- locked

  The *Snmp* component is locked.

- unlocked

  The *Snmp* component is unlocked.

The *Snmp* component may assume the following usage states:

- idle

  The SNMP agent is unable to send traps or handle requests. It is either locked or disabled or is just starting up.

- active

  The SNMP agent is able to handle traps and requests.

The table, "Valid combinations of the SNMP agent OSI states" (page 184), describes the valid combinations.

Table 42
Valid combinations of the SNMP agent OSI states

State combination (Administrative, Operational, Usage)	Details
Unlocked, Enabled, Active	The SNMP agent is fully operational, and able to process requests and send traps.
Unlocked, Disabled, Idle	The transport layer is down and/or the *VirtualRouter* is locked. Requests are not processed and traps are not sent out. Traps are not queued internally, and are discarded.

(Sheet 1 of 2)

Table 42 (continued)
Valid combinations of the SNMP agent OSI states (continued)

State combination (Administrative, Operational, Usage)	Details
Locked, Enabled, Idle	The SNMP agent is administratively prohibited from processing requests or sending traps. Traps are not queued internally, and are discarded.
Locked, Disabled, Idle	The SNMP agent is administratively prohibited from functioning. The transport layer is down and/or the *VirtualRouter* is locked. Requests are not processed and traps are not sent out. Traps are not queued internally, and are discarded.

(Sheet 2 of 2)

Polling Enterprise MIBs

Of primary concern to a manager is the *state* of individual network resources. Passport uses the OSI state attributes defined in ITU-T Recommendation X.731 | ISO 10164-2 *Open Systems Interconnection - Systems Management - Part 2: State Management Function* to model the state of Passport components.

Not all Passport components have OSI state, and of those that do, not all attributes support all states. For more information on the Passport implementation of OSI state, refer to 241-5701-520 *Passport 7400, 15000, 20000 Troubleshooting and Testing*.

OSI state of Passport components

Passport components maintain OSI state variables. These variables describe consistently the different aspects of the state of components. A scalable solution for polling those OsiState tables is provided through state summary tables.

Three state attributes and six qualifying status attributes are used. The standard also prescribes guidelines on their use, and state diagrams indicating legal state combinations and transitions between them. These attributes are represented as variables in the Enterprise MIB.

The state variables are

- *AdminState* represents the administrative state. This variable reflects whether or not the manager has the component turned on (locked, unlocked or shutting-down).
- *OperationalState* represents the operational state. This variable reflects whether or not the component is broken (disabled or enabled).
- *UsageState* represents the usage state. This variable reflects whether or not the component is in use (idle, active or busy).

The administrative and operational states are independent. For example, a component can be unlocked but disabled. Another component could be locked and enabled.

The status variables are

- *AvailabilityStatus* qualifies operational state, giving information about the availability of the resource.
- *ProceduralStatus* qualifies operational state, indicates the status of the object with respect to some procedure.
- *ControlStatus* qualifies administrative state, gives more details about administrative control over the object.
- *AlarmStatus* qualifies operational state, indicates whether an object has emitted an alarm that has not been cleared.
- *StandbyStatus* indicates the current role an object is playing when it is participating in a backup relationship.
- *UnknownStatus* if this attribute is true, the real state of the resource is unknown, and the values of the state attribute may not reflect the resource's actual state.

  Note: In the Enterprise MIB, OSI state information is always maintained in the *StateTable*. You should regularly poll only the *AdminStatus* and *OperationalStatus* variables from the *StateTable* for components to be monitored using a graphical network display. The other variables may be polled when an exception condition is detected.

For more information on how SNMP and OSI states interrelate, refer to "Relationship between SNMP and OSI states" (page 233).

Passport State Summary Table

The State Summary Table can be used by management applications to significantly reduce the amount of polling required for surveillance.

The Passport State Summary Table is a table of summarized state change information in the form of rolled-up state variables for each component class. The rolled-up state variables include

- a time stamp to indicate when the last time the state summary table was updated
- a time stamp for each component class to indicate the last time an instance of the component class experienced a state change
- the number of components that are down for each component class
- the number of components that are troubled for each component class

Instead of polling the state for all instances of component classes every poll cycle to determine if there was a change in OSI state or status, you need only poll the time stamp variables of the State Summary Table.

A further reduction in polling is provided by the time stamp which indicates when the last time the state summary table was updated. This time stamp is updated when any component class experiences a state change.

State Summary Table behavior

The following components are considered to be down:

- when a feature is loaded that contains component classes which support rolled-up state variables, an entry is added to the State Summary Table
- any state change will be propagated to the State Summary Table within five seconds of the event which triggered the state change
- when a CP switchover occurs, the last change time stamp for the state summary table and the last change time stamps for each component class is updated

- when a component class instance is added or deleted, the State Summary Table entry for the component class is updated within five seconds of activation completion

The following events trigger updating of the timestamp field:

- if a logical processor fails then all the instances of component classes on that processor are counted as being “down”
- if a component class instance is provisioned but not physically present then the instance is counted as being down
- if a component class instance has an administrative state of locked or an operational state of disabled then the instance is counted as being down

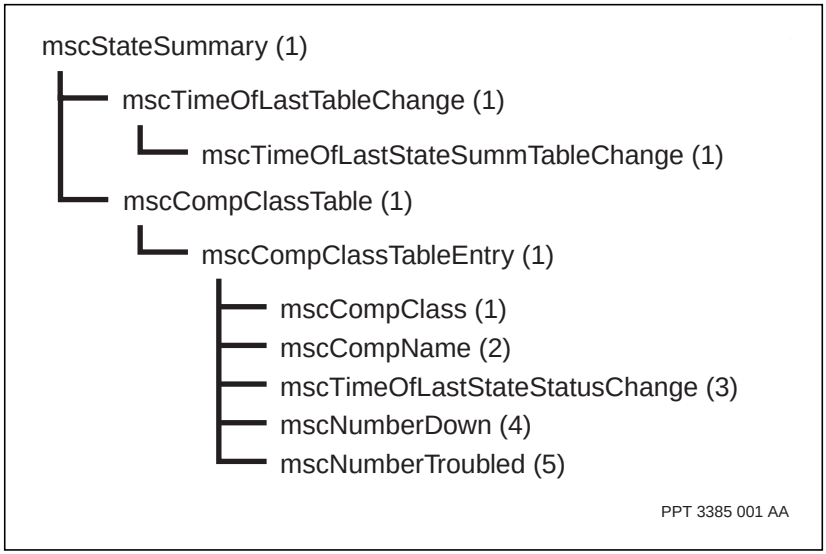
Components are considered troubled for the following reason:

- if a component class instance has either an availabilityStatus of degraded, an administrative state of shutting down or outstanding major/ critical alarms then the instance is counted as being troubled

Passport State Summary MIB

The Passport State Summary Table is defined in the MIB module *Nortel-MsCarrier-MscPassport-StateSummaryMIB*. The OBJECT IDENTIFIER tree structure for this MIB is illustrated in the figure “Passport State Summary MIB” (page 189). The State Summary table is indexed by the object identifier of a component class.

Figure 32
Passport State Summary MIB



Variable definitions

The table, “Passport state summary MIB variable definitions” (page 190), provides information on Passport state summary MIB variable definitions.

Table 43
Passport state summary MIB variable definitions

Attribute name	Type	Values	Variable Description
mscTimeOfLastState SummChange	TimeTicks	Hundredths of a second	The value of sysUpTime at the time that an entry in the State Summary Table detected a change
mscCompClass	OBJECT IDENTIFIER	1.3.6.1.4.1.562.36. 2.1.xxx where xxx is the OID of the component class	The object identifier which points to the SNMP MIB Arc for that particular component class. For example, a replication of 1.3.6.1.4.1.562.36.2.1.12(iso.org. dod.internet.private.enterprises. nortel.msCarrier.mscPassport. mscComponents.mscLp) corresponds to the mscLp component class.
mscCompName	DisplayString	Ascii	The string representation of the component class object identifier. For example, 1.3.6.1.4.1.562.36.2.1.12 (msCarrier.mscPassport. mscComponents.mscLp) is represented as the string, LogicalProcessor.
mscTimeOfLast StateStatusChange	TimeTicks	Hundredths of a second	The value of sysUpTime when an OsiState or OsiStateStatus change was detected for the component class. Currently monitored attributes include: AdminState, OperationalState, AvailabilityStatus and AlarmStatus.

(Sheet 1 of 2)

Table 43 (continued)
Passport state summary MIB variable definitions

Attribute name	Type	Values	Variable Description
mscNumberDown	Gauge	0 to 4294967295	The number of component instances that are down. A component is considered down when its administrative state is locked or its operational state is disabled
mscNumberTroubled	Gauge	0 to 4294967295	The number of component instances that are troubled. A component is considered troubled when its administrative status is shuttingDown or its alarm status is not empty or its availability status is degradedRepresents the interface's maximum transmission unit.

(Sheet 2 of 2)

Supported component classes

Not all Passport component classes support rolled-up state variables. The following lists the component classes that support rolled-up state variables:

- AtmInterface
- DPN Gateway
- FileSystem
- FrameRelayAtm
- FrameRelayNni
- FrameRelayUni
- LogicalProcessor
- LogicalProcessor DS1
- LogicalProcessor DS3
- LogicalProcessor E1
- LogicalProcessor E3

- LogicalProcessor Ethernet
- LogicalProcessor OamEthernet
- LogicalProcessor Eth 100Bt
- LogicalProcessor Hssi
- LogicalProcessor IlsForwarder
- LogicalProcessor InverseMultiplexerAtm
- LogicalProcessor JT2
- LogicalProcessor Sdh
- LogicalProcessor Sonet
- LogicalProcessor V35
- LogicalProcessor X21
- Shelf Card
- Trunk

Using Passport State Summary Table

When using the State Summary Table, you should either model component class instances or model the rows in the State Summary Table.

To model component class instances:

1 Perform a complete statewalk of the instances of a component class to populate the database.

2 Record a snapshot of the last change time stamp for the State Summary Table and a snapshot of the last change time stamp for each component class in the State Summary Table.

3 At regular intervals, poll the last change time stamp for the State Summary Table to determine if there has been any change to the State Summary Table.

4 If the State Summary Table has changed, poll the last change time stamp of each component class to determine which component classes have changed.

5 For each changed component class, poll instances of that component class to update the database.

To model rows in the State Summary Table:

1 Poll each row in the State Summary Table, and model each of them in the database.

2 Record a snapshot of the last change time stamp for the State Summary Table and a snapshot of the last change time stamp for each component class in the State Summary Table.

3 At regular intervals, poll the last change time stamp for the State Summary Table to determine if there has been any change to the State Summary Table.

4 If the State Summary Table has changed, poll the last change time stamp of each component class to determine which component classes have changed.

5 For each changed component class, poll the respective State Summary Table row and update the database.

A network operator would initiate a poll of instances of particular component classes if applicable specific troubleshooting is required.

Polling when LP is down

This section describes the responses that will be received when requesting *RowStatus* and Operational Data from components residing on an *Lp* that is down.

The *RowStatus* variable is used to indicate whether or not the component has been activated or not. Specifically, the *RowStatus* variable for each component will have the following behaviors:

The only valid values returned on a *get* command of RowStatus will be

- *notInService* - If the component has not been activated due to the *Lp* being down or an activation error. When in this state, a *get* command on variables in operational tables for this component will return zero/null/invalid values. The exceptions to this include *mscShelfCardRowStatus* and *mscLpRowStatus*. The *mscShelfCardRowStatus* table is always active regardless of whether the card is inserted or not. Use the

mscShelfCardInsertedCardType or *mscShelfCardAvailabilityStatus* to determine if the card is inserted. The *mscLpRowStatus* is always active. Use *mscLpActiveCard* to determine if it is up and active.

- *active* - If the component has been activated by the local management system. When in this state, a *get* command on variables in operational tables for this component will return values from the component's application. The *OperationalState* and *AdminState* should be polled to determine if the component is actually operational.

When a component is not activated (because the *Lp* on which it resides is down), a *get* command on variables in operational tables for this component will return zero/null/invalid values. See the table "Null values for operational tables" (page 194).

Table 44
Null values for operational tables

Type of Table	Value Description
Counter, Gauge, Integer	0
Enum	out of range ***Note:*** This value will be large to allow for growth of enum values.
String	** (null string)
IPAddress	0.0.0.0
Object Identifier	0.0
Array, Vector	These are statically sized and indexed tables. All of their elements will contain the appropriate zero/null/invalid values.
List, Replicated	These are dynamically sized and indexed tables. These rows will be removed as they should be.

Surveillance procedures using Enterprise MIB

This section provides information on how to perform various procedures related to SNMP surveillance using the Passport Enterprise MIB, including:

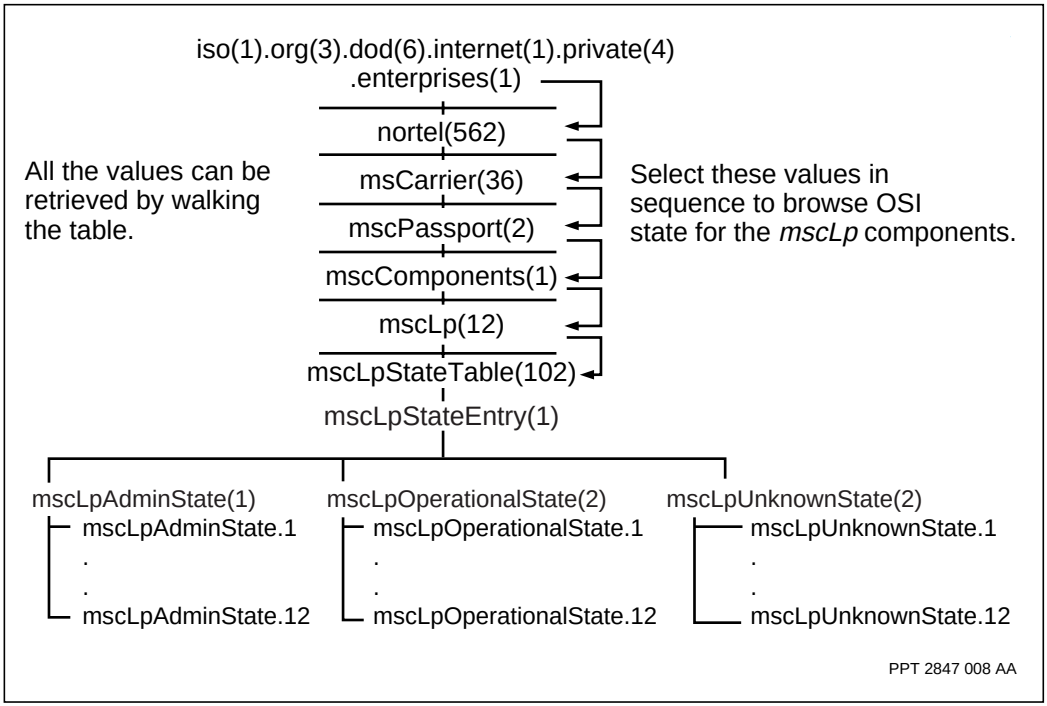
Browsing

Browsing the Passport Enterprise MIB is a good way to become familiar with its structure. The MIB has been designed to make browsing easy.

Browsers operate by allowing you to navigate through the MIB structure. This is done by selecting an OBJECT IDENTIFIER value and requesting that the next level down (the subordinate values) be displayed.

The figure, "Browsing the MIB" (page 196), illustrates the selections required to examine the *mscLpStateTable*. Most MIB browsers allow you to walk a table once you have navigated to its position in the tree.

Figure 33
Browsing the MIB



Understanding table walks

The names of SNMP variables are OBJECT IDENTIFIER values organized into a tree structure. A *getNext* request returns the value of the variable with the lexicographically next greatest name. A series of *getNext* requests can be used to walk or sweep a table, meaning: retrieve all of the values available in that table from the node.

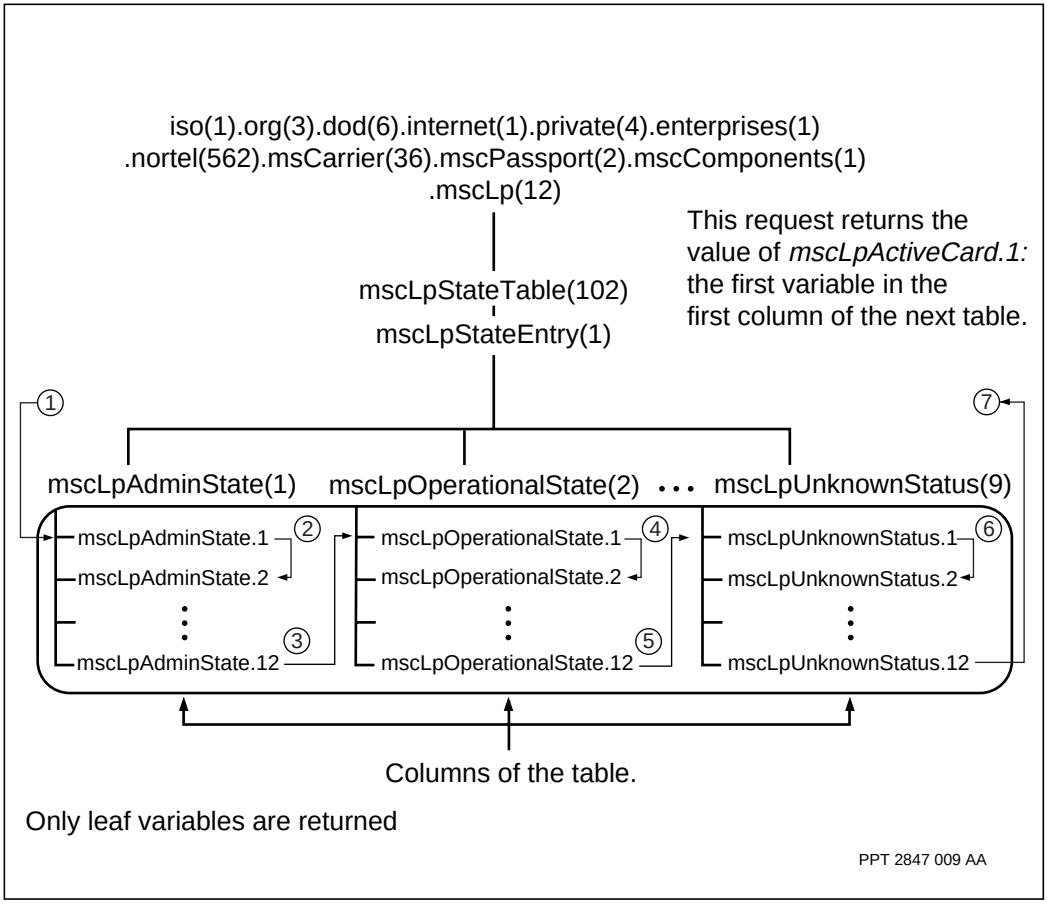
Column-by-column table walk

Walking a table column-by-column involves transmitting a series of *getNext* requests containing a single variable. This is the way most MIB browsers work. The figure, “Walking a table column-by-column” (page 197), illustrates a column-by-column table walk. The leaves of the tree are arranged horizontally in the figure to mimic the corresponding table structure.

The following steps are involved in walking a table column-by-column:

Note: Retrieving a table column-by-column is not very efficient: 109 requests are required for a table of nine columns with 12 rows.

Figure 34
Walking a table column-by-column



The following steps are involved in walking a table column-by-column:

1 A getNext request is launched starting at *mscLpStateTable*. Starting at *mscLpStateEntry or mscLpAdminState* would yield the same results:

```
getNext( mscLpStateTable )

--> mscLpAdminState.1= unlocked
```

2 A getNext request for any variable other than one at the bottom of a column returns the next row's value. For example:

```
getNext( mscLpAdminState.1 )

--> mscLpAdminState.2= unlocked
```

Iterating on the returned value results in the return of the entire *mscLpAdminState* column.

3 A getNext request for a variable at the bottom of any but the right-most column returns the first value in the next row. For example:

```
getNext( mscLpAdminState.12 )

--> mscLpOperationalState.1= enabled
```

4 The *mscLpOperationalState* column is retrieved in the same way the *mscLpAdminState* column is retrieved in step 2.

5 Again, the request moves to the next column. This continues until the last column.

6 The last column is retrieved in the same way as the previous, described in steps 2 and 4.

7 A getNext request for a variable at the bottom of the right-most column returns the first value in the next *table*. For example:

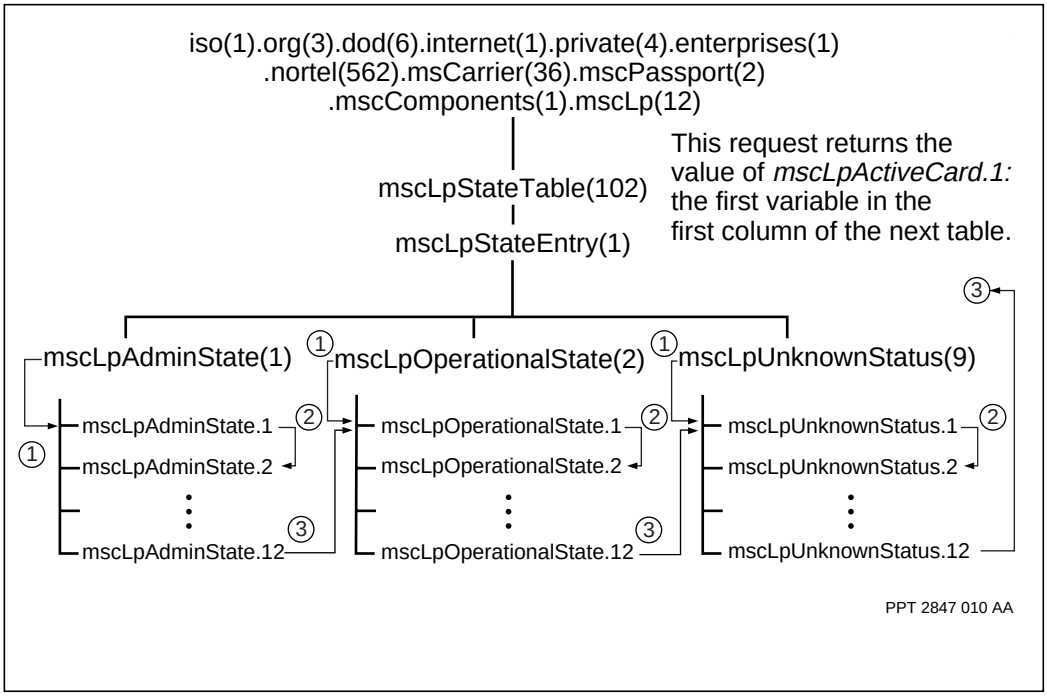
```
getNext( mscLpUnknownStatus.12 )

--> mscLpActiveCard.1=1.4.3.2.562...
```

Row-by-row table walk

Walking a table row-by-row involves transmitting a series of *getNext* requests containing one variable from each column. Applications designed to retrieve entire tables should use this approach because it provides better performance. The figure, "Walking a table row-by-row" (page 199), illustrates a row-by-row table walk. The leaves of the tree are arranged horizontally in the figure to mimic the corresponding table structure

Figure 35
Walking a table row-by-row



The following steps are involved in walking a table row-by-row:

1 A getNext request is launched starting at the top of all of the columns:

`getNext( mscLpAdminState, mscLpOperationalState, ..., mscLpUnknownStatus )`

```
--> mscLpAdminState.1= unlocked,
mscLpOperationalState.1=enabled, ...,
mscLpUnknownStatus.1=false
```

2 A getNext request for variables other than those at the bottom of a column returns the next row's values. For example:

`getNext( mscLpAdminState.1, mscLpOperationalState.1, ..., mscLpUnknownStatus.1 )`

```
--> mscLpAdminState.2=unlocked,
mscLpOperationalState.2=enabled, ...,
mscLpUnknownStatus.2=false
```

Iterating on the returned values results in the return of the entire table.

3 A getNext request for a variable at the bottom of any but the right-most column returns the first value in the next row. A getNext request for a variable at the bottom of the right-most column returns the first value in the next *table*. For example:

```
getNext( mscLpAdminState.12,
mscLpOperationalState.12, ...,
mscLpUnknownStatus.12 )
```

```
--> mscLpOperationalState.1=enabled,
mscLpOsiUsage.1=active, ...,
mscLpActiveCard.1=1.4.3.2.562...
```

Note: Retrieving a table row-by-row is efficient: 13 requests are required for a table of nine columns with 12 rows.

GetBulk request

To further minimize the number of protocol exchanges that occur when transferring large files, you can use a *getBulk* request. Whereas a *getNext* request returns the value of only the next instance of an object or objects, a *getBulk* request returns the value of the next <n> instances of an object or objects.

Note: A *getBulk* request is only accessible if communicating with a v2c manager and when the *supportedVersions* attribute is set to include v2c. If *supportedVersions*, an attribute of the Snmp component, is not set to include v2c and a getBulk message is received from a v2c manager, the *inBadVersions* counter increments by 1.

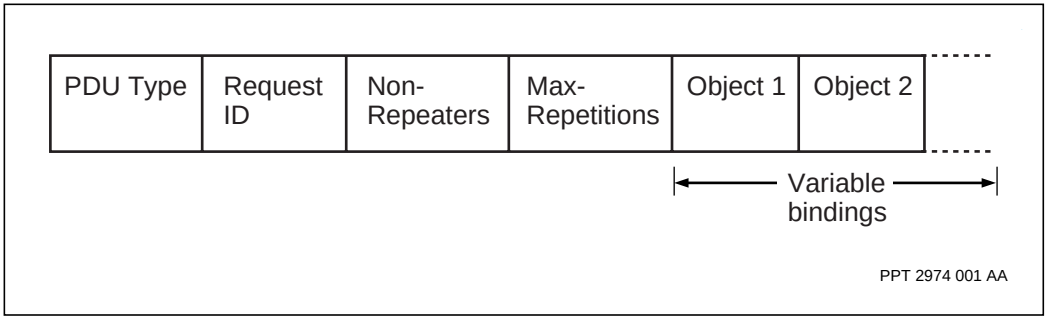
The number of variables returned is determined by the variable bindings requested and the values in the non-repeaters field and the max-repetitions field.

The non-repeaters field allows the manager to specify the number of variable bindings that require a single response. For example, a value of two in this field indicates that the first two variable bindings will have a single value returned.

The max-repetitions field indicates to the agent the number of objects and their values that are returned. For example, a value of three in this field indicates that three rows of the table are returned.

The figure “GetBulk request PDU” (page 201) displays the format of a *getBulk* request PDU.

Figure 36
GetBulk request PDU



Optimizations

The Passport Enterprise MIB has been optimized based on anticipated requirements of management applications. You may take advantage of these optimizations to improve the efficiency of your applications and the execution of regular operations activities.

The Passport agent has been optimized to return rows of tables as opposed to individual variables. You should not walk a table column-by-column if you are able to walk the table row-by-row.

As described in “Attributes and attribute groups” (page 78) there are two kinds of attributes: provisionable and operational.

The Passport node is a multi-processor node. Provisioning data, which changes only as a result of operator intervention, is co-located on the control processor with the SNMP agent. This means that access to this data is faster. This is important because applications using provisioning data typically require sporadic retrieval of large amounts of data in response to operator requests against graphical user interfaces. Here are some guidelines for optimizing retrieval of provisioning data:

- Retrieve only provisioning data in the request. Mixing requests for operational and provisional data slows down the retrieval of the provisioning data.

- Requests for data located in the same instance subtree, for example the *mscFrUni.1* subtree, is slightly faster than requests for data from different instance subtrees.
- Use a smaller number of larger PDUs rather than a larger number of smaller PDUs.

Operational data is distributed across a node's functional processors. Retrieval of data from different functional processors is slower than from a single functional processor. For example, retrieving a row of an operational table with five variables is much faster than retrieving the same operational variable from five different components located on different functional processors.

Uses of the ifTable

The *ifTable* is used by managers as a universal facility for fault and performance management. For fault management, managers perform the following operations:

- Listen for *linkUp* or *linkDown* traps then poll *ifOperStatus*
- Check the value of *ifOutQlen* for congestion.
- Compare traffic versus bandwidth for congestion.
- Examine rates of change of various counters, such as the in and out octets, in and out errors, and in and out packets.

For performance management, managers perform the following operations:

- Use *linkUp* and *linkDown* traps to determine availability.
- Poll and graph Counters to chart utilization.
- Poll and log Counters for post-processing and analysis.

Since the *ifTable* is available on all systems, managers can use simple, common algorithms to manage different classes of equipment provided by different vendors. All of the Open Management Platforms provide simple tools to facilitate this, such as generic graphing and logging tools, and simple threshold event tools.

In essence, the *ifTable* provides a suite of enabling capabilities familiar to, and used by, most managers. You will find related information in “Relationship between SNMP and OSI states” (page 233).

An example of a third-party configuration

If you are using HP OpenView or IBM NetView to monitor Passport, the contents of the Passport Enterprise Traps is automatically formatted to appear in the event display. The formatting includes the mapping of enumerations to their labels, and formatting of the *eventTime* parameter.

If you are using HP OpenView or IBM NetView on its own, then you must use the event configuration tool to define a format for the traps.

The following is an example of how to configure HP OpenView’s events to receive traps from Passport nodes.

Note: For this example, HP OpenView version NNM 3.3 was used.

1. From Options menu bar, select <Event Configuration>.
2. When the event configuration appears, add an Enterprise for Passport Alarms.

   `Enterprise Name: PassportAlarms`

   `Enterprise ID:.1.3.6.1.4.1.562.36.2.1.3.2`
3. Add an Event for each of the traps representing Passport Alarms (critical (*mscCriticalAlarm(1)*), major (*mscMajorAlarm(2)*), minor (*mscMinorAlarm(3)*), warning (*mscWarningAlarm(4)*), cleared (*mscClearedAlarm(5)*), and indeterminate (*mscIndeterminate(6)*)).
4. Specify the Enterprise name and ID for Nortel Networks’ Enterprise alarmTrap (.1.3.6.1.4.1.562.36.2.1.3.2), then confirm the selection.
5. Enter the Event Name (for example, <Critical>), and the Enterprise specific trap number (for example, <1>).
6. Enter Passport Critical Alarm for Event Description.
7. Set the Event Category as desired.
8. Set the Severity to *critical*.
9. Configure the Event Log Message to display the variables you require from the mandatoryAlarmInfo group defined in the *Nortel-MsCarrier-*

MscPassport-AlarmMIB. The following information will help you to determine which variables to include:

The *mscComponentRowPointer* variable should be selected if you will be invoking a MIB browser to query the variables of the component which issued the alarm.

The *mscComponentName* variable should always be selected.

The tool may not be able to format the *mscEventTime* variable which uses the DateAndTime textual convention. In that case, it may be best to exclude the *mscEventTime* variable from the message and rely on the timestamp that the tool automatically puts into the message.

The tool may not be able to map enumerated types to their labels. For example, the *mscActiveListStatus* value would appear as 0,1 or 2 not as message, set or clear. If this is the case, the *mscActiveListStatus* variable should be selected. The *mscSeverity* is already identified by the specific trap number and need not be selected. The *mscAlarmType* and *mscProbableCause* variables could be excluded from the display at the operator's discretion since the *mscNtpIndex* and *mscCommentData* may be sufficient. If they are not included the network operator would have to refer to the *Nortel-MsCarrier-MscPassport-AlarmMIB* to see the numeric mapping to enumeration.

The *mscNtpIndex* and *mscCommentData* should be selected.

For example, the initial format string, when changed from

```
Trap: generic $G specific: $S args ($#): S*
```

to:

```
$2, Status(0-Msg,1-Set,2-Clr) $4, NTP Index $8,
Comment: $9
```

would result in a Passport trap being formatted as:

```
CRITICAL Sun Mar 02 14:28:59 1.2.3.4 EM/NODER72
ShelfCard/2,
Status (0-Msg, 1-Set, 2-Clr) 1, NTP Index 70120100,
Comment: Card is Disabled
```

10 Repeat the configuration steps until all the events are configured.

11 Close the Event Configuration window by selecting <OK>.

12 Ensure the Passport Enterprise Traps are being displayed in the SNMP Events window according to the format you defined.

Logging of SNMP Set Requests

All non-passive SNMP commands (set requests) issued to Passport can be captured in the existing data collection system (DCS) log stream. There are several benefits to logging SNMP set requests relating to security, system integrity, fault management, and troubleshooting.

The log information is formatted in such a way that it can be quickly and easily interpreted by network operators. The log format includes the following fields:

- date and time of the SET request
- IP address of the SNMP manager that issued the request
- actual management information base (MIB) variables and related values that were set
- CLI equivalent of the commands which are issued. This field is only available in the setting of Enterprise MIB variables.
- response message which will either indicate that the set request was successful or, in the case of a failure, include an error message indicating a reason for the failure

  Note: The variable bindings specified in the SNMP set request are logged as MIB object names and instances.

Each field in a log record is limited to a fixed size of 255 characters, with the exception of the response field (275 characters). SNMP set requests exceeding this limitation will result in multiple logs, however individual varbinds exceeding this limit will be truncated.

Note: The response to the SNMP request is only included in the last record. For more information on the response field, see “Detailed error response to a set request” (page 246).

Logging is either turned on or off for all applicable Passport interfaces, therefore, there is no separate control of SNMP logs. For more information about DCS log streams, see 241-5701-611 *Passport 7400, 15000, 20000 Data Collection Guide*.

Displaying SNMP logs

You can display SNMP logs using the local operator or a telnet session from the CLI interface by entering the following command:

```
set nmis <w> session/<x> dataStreams log
```

where:
<**w>** is local or telnet, and <x> is the number of the session.

Chapter 9
Passport verbs and SNMP

Passport verbs allow an operator to perform a specific action on a component. This chapter contains information on how Passport verbs can be invoked via SNMP

Currently, only the following Passport verbs are supported via SNMP:

- The commit verb (only applicable under the Passport provisioning component). Refer to “Invoking the commit verb” (page 207) for details.
- The lock verb (only applicable under the Passport *FrUni Dlci* component). Refer to “Invoking the lock verb” (page 209) for details.
- The unlock verb (only applicable under the Passport *FrUni Dlci* component). Refer to “Invoking the unlock verb” (page 210) for details.

For more information on using Passport verbs, refer to 241-5701-050 *Passport 7400, 15000, 20000 Commands*.

Invoking the commit verb

This section describes how to use the SNMP *set* request to invoke the commit command.

To invoke the commit command:

1 The SNMP manager must issue a *set* request on the *mscProvCommitFileOption* variable and give the variable a valid Passport provisioning file name (for example “DLZNode_July23_2000”).

Note: Only the *mscProvCommitFileOption* variable can be specified in a SNMP *set* request to successfully invoke the commit command. The inclusion of any other variables causes the SNMP *set* request to fail.

2 The SNMP agent processes the SNMP *set* request from the SNMP manager, ensuring that:

— the request is received from an authorized SNMP manager

— the request is received from a manager with access to the *mscProvCommitVerbTable*

— the file name provided in the *mscProvCommitFileOption* variable that executes the commit command is valid

If all three checks are valid, the commit command is invoked. To ensure if the commit command has actually completed processing, a separate activity is required. Refer to "Confirming the status of the configuration file" (page 208) to determine if the commit command has completed processing.

3 Once the commit command is invoked, a *set* response is returned to the requesting SNMP manager, stating that the commit command has been successfully invoked.

Note: While the commit command is processing, no other SNMP set requests or configuring through the CLI can be done until processing of the commit command is complete. This may result in a delay of several minutes, with larger configurations taking longer.

Confirming the status of the configuration file

Since the *set* request does not guarantee that the commit command has been invoked, it is recommended that a check be made to determine if the configuration file has been successfully committed.

To determine the status of the configuration file:

1 Perform a SNMP *get* request on three variables:

— *mscProvCommittedFileName*, which represents the committed file currently on the node

— *mscProvProvisioningActivity*, which represents the command currently being processed

— *mscProvActivityProgress*, which represents the progress of the current command

2 The commit command is still processing if:

— the *mscProvCommittedFileName* variable returns the value of the committed file name prior to invoking the commit command

— the *mscProvProvisioningActivity* variable returns a value of "committing"

— the *mscProvActivityProgress* variable returns a percentage value (for example, "35% done")

3 The commit command is considered to be finished processing when:

— the *mscProvCommittedFileName* variable returns the value of the file name specified in the SNMP *set* request

— the *mscProvProvisioningActivity* variable returns a value of "none"

— the *mscProvActivityProgress* variable returns a value of "n/a"

Note: If the value of the *mscProvCommittedFileName* variable is not the exact file name specified in the SNMP *set* request, the commit command has failed.

Invoking the lock verb

This section describes how to use the SNMP *set* request to invoke the lock command.

Note: The lock verb can only be invoked via SNMP under the *FrUni Dlci* component.

To invoke the lock command:

1 The SNMP manager must issue a *set* request on the *mscFrUniDlciLockNoOptionsFlag* variable, setting the value to 1.

Note: Setting the variable to any value other than 1 will not invoke the lock command.

2 The SNMP agent processes the SNMP *set* request from the SNMP manager, ensuring that:

— the request is received from an authorized SNMP manager

— the request is received from a manager with access to the *mscFrUniDlciLockVerbTable*

— the value of the *mscFrUniDlciLockNoOptionsFlag* variable is set to 1

— the instance of the specified FrUni and Dlci exists on the Passport

If all four checks are valid, the lock command is invoked on the specified FrUni Dlci link. To ensure if the lock command has actually completed processing, a separate activity is required. Refer to "Confirming that the FrUni Dlci link is locked" (page 210) to determine if the lock command has completed processing.

3 Once the lock command is invoked, a *set* response is returned to the requesting SNMP manager, stating that the lock command has been successfully invoked.

Confirming that the FrUni Dlci link is locked

Since the *set* request does not guarantee that the lock command has been invoked, it is recommended that a check be made to determine if the lock command has been successfully completed.

To determine the status of a lock command:

1 Perform a SNMP get request on the *mscFrUniDlciAdminState* variable, which represents the OSI administration state of a specific FrUni Dlci link.

2 If the *mscFrUniDlciAdminState* variable returns a value of "locked", the lock command has successfully finished processing.

Invoking the unlock verb

This section describes how to use the SNMP *set* request to invoke the unlock command.

Note: The unlock verb can only be invoked via SNMP under the *FrUni Dlci* component.

To invoke the unlock command:

1 The SNMP manager must issue a *set* request on the *mscFrUniDlciUnlockNoOptionsFlag* variable, setting the value to 1.

Note: Setting the variable to any value other than 1 will not invoke the unlock command.

2 The SNMP agent processes the SNMP *set* request from the SNMP manager, ensuring that:

— the request is received from an authorized SNMP manager

— the request is received from a manager with access to the *mscFrUniDlciUnlockVerbTable*

— the value provided in the *mscFrUniDlciUnlockNoOptionsFlag* variable is set to 1

— the instance of the specified FrUni and Dlci exists on the Passport

If all four checks are valid, the unlock command is invoked on the specified FrUni Dlci link. To ensure if the unlock command has actually completed processing, a separate activity is required. Refer to “Confirming that the FrUni Dlci link is unlocked” (page 211) to determine if the unlock command has completed processing.

3 Once the unlock command is invoked, a *set* response is returned to the requesting SNMP manager, stating that the unlock command has been successfully invoked.

Confirming that the FrUni Dlci link is unlocked

Since the *set* request does not guarantee that the unlock command has been invoked, it is recommended that a check be made to determine if the unlock command has been successfully completed.

To determine the status of an unlock command:

1 Perform a SNMP *get* request on the *mscFrUniDlciAdminState* variable, which represents the OSI administration state of a specific FrUni Dlci link.

2 If the *mscFrUniDlciAdminState* variable returns a value of “unlocked”, the unlock command has successfully finished processing.

Chapter 10
Provisioning using SNMP set request

MIB variables can be set according to standard SNMP practices and following the SNMP all-or-nothing paradigm. Modifications are made permanent through invoking the commit command via a SNMP set request.

The time to perform the semantic check varies depending on

- the type of data changed.
- the size of the switch configuration.
- how busy the switch is.

In particular, semantic checking of changes to the *Lp* or *Software* hierarchies can be quite long.

> ***Note:*** It is recommended that SNMP *sets* on the *Lp* or *Software* hierarchies not be done, since the processing of a *set* command blocks the SNMP request processing.

You need to understand how provisionable SNMP *sets* interface with the on-switch provisioning system.

The Passport agent always interacts with the Passport views when performing *set, get, getNext, and getBulk* operations.

> ***Note:*** An important distinction must be made here between Passport provisioning views and the MIB views described under the Administrative Model. In this chapter, the term view will always refer to

a Passport provisioning view. The Passport provisioning system is described in detail in 241-5701-045 *Passport 7400, 15000, 20000 Management System User Interface Guide*.

SNMP *get*, *getNext*, and getBulk requests always return data from the current view. SNMP *set* requests on operational data also apply to the current view.

SNMP *set* requests on provisionable data are applied directly to the edit view, semantically checked, activated and confirmed. The configuration changes are committed to become the current view and saved as the boot configuration once a SNMP set request invoking the commit command is issued successfully.

An SNMP *set* request fails to execute when

- the edit view is already open. This means that some other manager is provisioning the system, possibly using the local console, a telnet session, or a FMIP interface.
- the edit view is not open but differs from the current view. This means that some other manager is in the process of provisioning the system.
- a complete semantic check of the entire edit view is required. The first semantic check after a software upgrade is always a complete semantic check. A complete semantic check is also required if the last activation provisioned a feature on an *Lp* which caused additional software to be loaded on the control processor.

Successfully executing an SNMP *set* request will

- open the edit view.
- perform, in the edit view, modifications to provisioning data indicated by the request. This process fails if any of the modifications are syntactically incorrect.
- perform a semantic check. The duration of the semantic check varies depending on the type of data changed and the size of the switch configuration.
- activate and close the edit view.
- confirm the change.

Changes to provisionable data made using SNMP set requests become committed as the boot configuration once the commit command is invoked via a separate SNMP set request.

Adding and deleting rows

Some types of components can be added and deleted. This is done by using the *RowStatus* variable in the *RowStatusTable*.

Adding a component is accomplished by setting the *RowStatus* variable in the *RowStatusTable* to createAndGo, where the indicated component does not already exist.

Deleting a component is accomplished by setting the *RowStatus* variable in the *RowStatusTable* to delete.

> ***Note:*** When adding some components, default subcomponents are added automatically. For example, adding a *FrUni* component automatically adds a *FrUni Framer* subcomponent.

The following rules apply when processing rowStatus variables:

- The *rowStatus* variable of an existing component cannot be set to createAndGo. The *set* request fails with a return code of genError.
- The *varBindList* cannot contain duplicate *rowStatus varbinds*. Duplicate *rowStatus varbinds* is equivalent to an attempt to create the same component twice. Such a *set* request fails with a return code of genError.
- A single *varbindList* can contain several unique *rowStatus* variables. This is equivalent to adding several different types and instances of components.
- The *rowStatus* variable of a non-existent component can always be set to delete.

In a SNMP *set* request, there is no restriction on the order or mix of *varbinds* which can appear in the *varBindList*. The SNMP Agent groups and processes the *varbinds* in the following order:

1 all *rowStatus varBinds*

2 changes to provisioned *varBinds*

3 changes to operational *varBinds*

The following rules apply when processing *set* requests on provisioned or operational *varbinds*:

- The component must exist.
- If a component is being added, a *rowStatus varbind* creating the component must be included in the *varBindList*.

Values of structured attributes

Values can be added and deleted from lists and replicated attributes.

Deleting a value from a list or replicated attribute is accomplished by setting the *RowStatus* variable for that value to *destroy*. If the indicated value does not exist, no error is returned. The desired result, the attribute does not contain the value, has been achieved.

For convenience, the *RowStatus* variable does not need to be used to add values to lists or replicated attributes. This can be done by simply setting the variable to the desired value. For example,

set mscFrUniListValue.1.12 12 —> add *12* to the list

set mscFrUniReplicatedValue.1.4 114 —> add *114* to the replicated attribute at index 4

If the specified value already exists for a list attribute, then no error is returned. If a value exists for the indicated index for a replicated attribute, the value is changed to the one presented.

A List index and its value must be consistent. The following *set* request fails with a *badValue* error:

set mscFrUniListValue.1.6 12 —> 12 does not match 6

Constraints

There are many Enterprise MIB variables that should not be changed through SNMP *sets*. These variables should be set from one of the other management interfaces, preferably when commissioning the node.

An SNMP *set* on a variable which, when activated, would cause the node to restart is not allowed and will therefore fail with a return code of genErr. For example, any change to the *mscModNodeName* or *mscModNamsId* will fail in this manner.

An SNMP *set* request which triggers the need for a complete semantic check will disable all future *set* requests. After this point, all *set* requests will immediately result in a return code of genError. No further SNMP *set* requests will be processed until the condition has been cleared, by performing a *check prov*, from one of the other management interfaces.

It is strongly recommended that you do not use SNMP *sets* to create or modify the following components:

- shelf and shelf card components
- logical processor and its subcomponents
- software and its subcomponents

A *set* request which disables the trunk or port used to access the Passport node, may cause the Passport to become isolated. Because the provisioning change is automatically confirmed, an intervention may be required to restore connectivity.

CAUTION
Loss of service
This operation may cause physical isolation of the Passport node.

Appendix A
Retrieving Standard MIBs

This appendix provides instructions on retrieving Standard MIBs.

Retrieving RFCs containing Standard MIBs

Request for comments can be obtained via E-mail or FTP from many RFC repositories. Most of these repositories also now have World Wide Web (WWW) servers. Two of the well-known RFC repositories are

- DS.INTERNIC.NET - InterNIC directory and database services
- ISI.EDU web site

InterNIC directory and database services

RFCs can be obtained from DS.INTERNIC.NET via FTP, Wide Area Information Server (WAIS), and electronic mail. Through FTP, RFCs are stored as *rfc/rfcnnnn.txt* or *rfc/rfcnnnn.ps* where *nnnn* is the RFC number. Login as anonymous and provide your e-mail address as the password. Through WAIS, you can use either your local WAIS client or Telnet to DS.INTERNIC.NET and login as *wais* (no password required) to access a WAIS client. Help information and a tutorial for using WAIS are available on-line. The WAIS database to search is *rfcs*.

Directory and database services also provides a mail server interface. Send an e-mail message to *mailserv@ds.internic.net* and include any of the following commands in the message body:

```
document-by-name rfcnnnn
```

where *nnnn* is the RFC number. The text version is sent

```
file /ftp/rfc/rfcnnnn.yyy
```

where *nnnn* is the RFC number and *yyy* is ‘txt’ or ‘ps’

```
help
```

gets information on how to use the mailserver

The InterNIC directory and database services collection of resource listings, Internet documents such as RFCs, FYIs, STDs, Internet drafts, and publicly accessible databases are also now available through Gopher. All collections are wais-indexed and can be searched from the Gopher menu.

To access the InterNIC Gopher servers, connect to:

internic.net port 70

contact: *admin@ds.internic.net*

ISI.EDU web site

The world wide web address is:

http://www.isi.edu/rfc-editor

For more information on where and how to get new RFCs, send an e-mail message to *rfc-info@ISI.EDU* with the following subject and body:

subject: *getting rfcs*

body: *help: ways_to_get_rfcs*

Appendix B
SNMP object identifier usage

This appendix describes SNMP OBJECT IDENTIFER usage for Passport Carrier products, including:

msCarrier SNMP registration requirements

There are a number of information objects associated with the development of SNMP managed systems which require registration (that is, require an OBJECT IDENTIFIER value be assigned):

- values of *sysObjectID* (see "System object IDs" (page 223))
- MODULE-IDENTITY statements
- AGENT-CAPABILITIES statements
- OBJECT-GROUP statements
- NOTIFICATION-GROUP statements
- objects (variables) and notifications defined within Enterprise MIBs

The Passport msCarrier object identifier tree

The figure, “Object identifier tree for Passport msCarrier registration” (page 222), illustrates the MIB structure used to register Nortel’s Carrier SNMP information objects under *{ enterprises nortel(562) msCarrier(36) }*. The *enterprises* arc is defined by the IETF as:

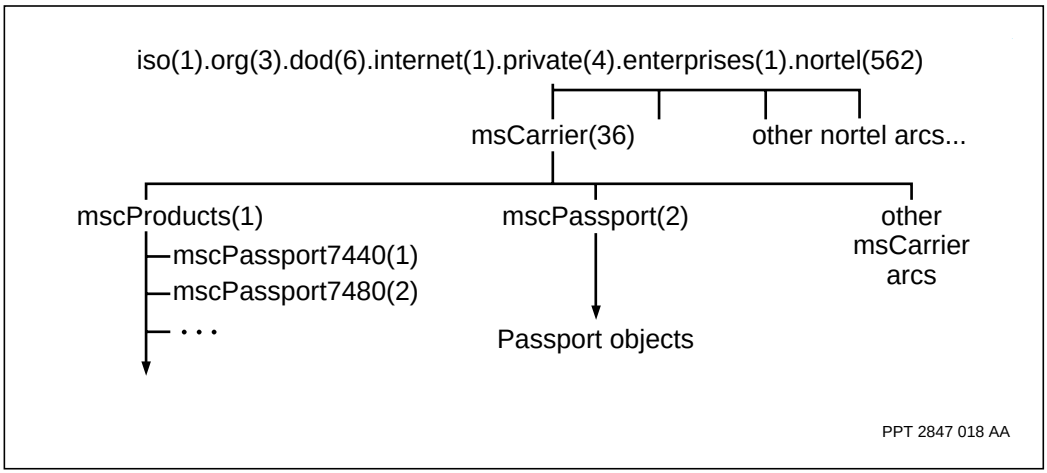
```
enterprises OBJECT IDENTIFIER ::= { iso(1) org(3)
dod(6)
internet(1) private(4) 1 }
```

The figure, “Object identifier tree for Passport msCarrier registration” (page 222), shows the arcs of concern to msCarrier registration:

- a single arc, *mscProducts(1)*, is used to assign values for *sysObjectID* for all *msCarrier* products
- a separate arc, *mscPassport(2)*, is used to assign values to *msCarrier* information objects

The values for these arcs are defined in the MIB module *Nortel-MsCarrier-MscPassport-UsefulDefinitionsMIB.*

Figure 37
Object identifier tree for Passport msCarrier registration



System object IDs

Underneath the *products* arc are values for the *sysObjectID* variable. These values identify different *msCarrier* products.

The SNMP variable *sysObjectID*, from the MIB II *system* group, is used to identify a resource. This value provides an easy and unambiguous means for determining what kind of box is being managed. A number of open management systems use the *sysObjectID* variable to display an appropriate Icon on the graphical user interface.

Values for *sysObjectID* are lexicographically smaller than those used for any variables in any msCarrier product. This insures that a getNext sweep can always begin using a value retrieved from *sysObjectID*.

The values for *sysObjectID* for msCarrier products are defined in the MIB module *Nortel-MsCarrier-MscPassport-UsefulDefinitionsMIB.* Some of these values are illustrated in the figure "Object identifier tree for Passport msCarrier registration" (page 222).

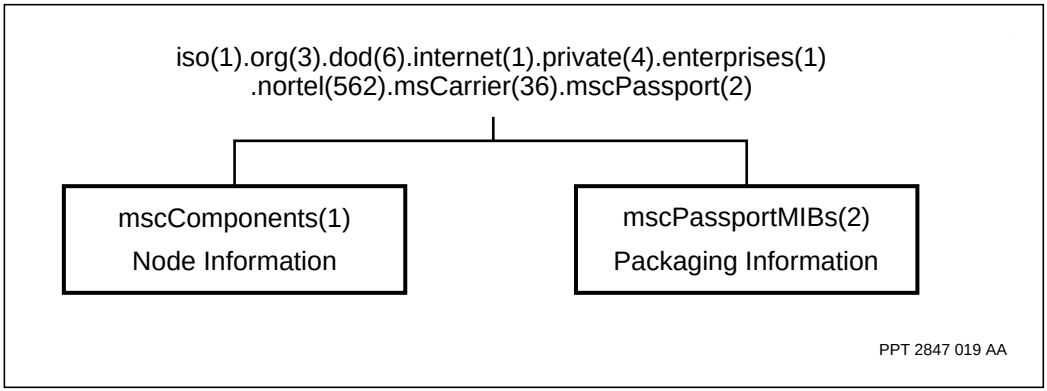
Components and MIB modules

Management Information for *msCarrier* is divided into two categories to facilitate MIB browsing independent of packaging:

- Node information is defined under the *mscComponents* arc.
- Packaging information is defined under the *mscPassportMIBs* arc.

The figure, "Object identifier tree for Passport msCarrier registration" (page 224), illustrates the MIB structure that is used to register these arcs. The values for these arcs are defined in the MIB module *Nortel-MsCarrier-MscPassport-UsefulDefinitionsMIB.*

Figure 38
Object identifier tree for Passport msCarrier registration



Components

The *{ mscPassport(2) mscComponents(1) }* arc is used to register top level components. A distinct arc (that is, *mscComponents*) is used to facilitate MIB browsing independent of packaging. All invocations of the SNMP OBJECT-TYPE and NOTIFICATION macros are registered in the subtree rooted at this arc, as described in “Passport Enterprise MIBs” (page 71).

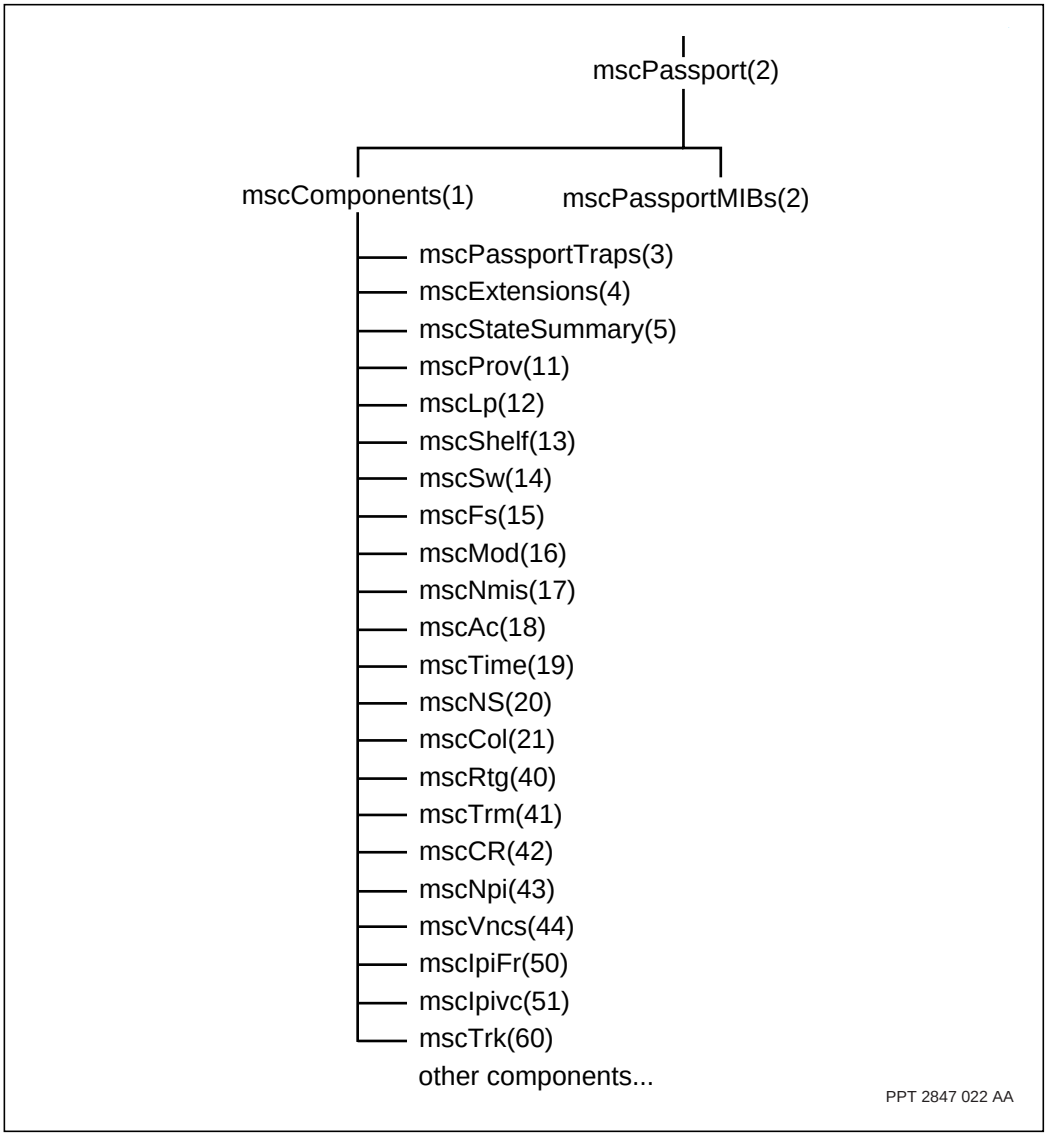
Always browse under *mscComponents* for information about the node.

There are a number of distinguished arcs reserved for special components:

- The arc (1) is reserved for the root component, should it need to be represented in the Enterprise MIB in the future.
- The *mscPassportTraps* arc represents a component concerning asynchronous events. Again, this is not a real component, but is modeled as such for registration.
- The *extensions* arc represents a component describing extensions to Standard MIBs, such as those for the MIB II *ifTable*. Again, this is not a real component, but is modeled as such for registration.
- Arcs (6) through (10) are reserved for future special components.

The figure, “Object identifier tree for msCarrier’s top-level components” (page 225), illustrates the registration of top-level components.

Figure 39
Object identifier tree for msCarrier's top-level components



Object-type and notification definitions

"Component model" (page 76) describes in detail the registration of all OBJECT-TYPE and NOTIFICATION definitions. These definitions appear in the different MIB modules as required.

MIB modules

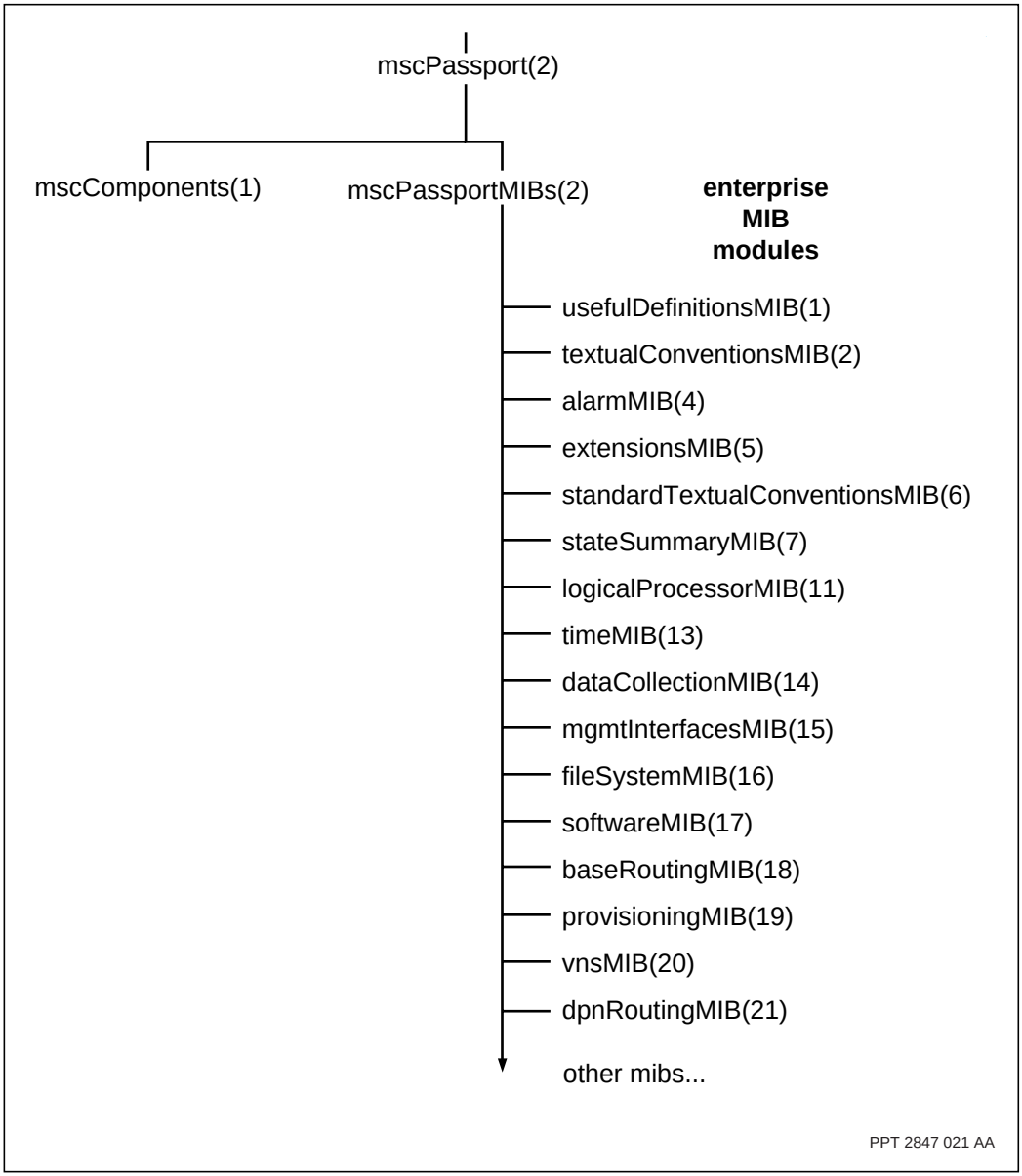
Under the *{ mscPassport(2) mscPassportMIBs(2) }* arc are separate arcs used to identify Enterprise MIB modules for either SMIv1 or SMIv2 format. These MIB Modules are aligned with msCarrier packaging: a feature may have 0, 1 or more associated MIB modules. All OBJECT-GROUP, NOTIFICATION-GROUP and AGENT-CAPABILITIES statements are registered in the subtree rooted at this arc.

There are a number of distinguished *mscPassportMIBs* representing special modules:

- *Nortel-MsCarrier-MscPassport-UsefulDefinitionsMIB* represents the upper reaches of the *msCarrier* MIB structure.
- *Nortel-MsCarrier-MscPassport-TextualConventionsMIB* represents the textual conventions specific to the *msCarrier* Enterprise MIB. This module is defined in "Textual conventions" (page 229).
- *Nortel-MsCarrier-MscPassport-AlarmMIB* provides a specification of *msCarrier* alarms.
- *Nortel-MsCarrier-MscPassport-ExtensionsMIB* provides extensions to certain Standard MIBs. For example, the MIB II *ifTable* has been extended.
- *Nortel-MsCarrier-MscPassport-StandardTextualConventionsMIB* provides a specification of all supported generic standard textual conventions. This is provided so *msCarrier* Enterprise MIB users do not have to search for, acquire, and compile the source Standard MIBs themselves (refer to "Surveillance" (page 183)).
- Arcs (8) through (10) are reserved for future special modules.

The figure, "Object identifier tree for msCarrier MIB modules" (page 227), displays the object identifier tree for the enterprise *msCarrier* MIB modules.

Figure 40
Object identifier tree for msCarrier MIB modules



Agent-capabilities and group definitions

"Passport Enterprise MIBs" (page 71) describes in detail the registration of all AGENT-CAPABILITIES statements and OBJECT-GROUP and NOTIFICATION-GROUP definitions, which are registered subordinate to the arcs assigned to the MIB modules containing them. These definitions appear in the different MIB modules as required.

Appendix C
Textual conventions

When designing a MIB module it is often useful to define new types similar to those defined in the Structure of Management Information (SMI). In comparison to a type defined in the SMI, each of these new types has a different name, a similar syntax, and more precise semantics. These newly defined types are termed Textual Conventions and are used for your convenience when reading MIB modules and as well by management applications to determine the display characteristics of new types. Textual conventions enhance the readability of MIB modules and ease comparisons with companion documents (for example, comparing a MIB with a protocol specification).

Variables defined using a textual convention are always encoded by means of the rules that define their primitive type. However, textual conventions often have special semantics associated with them, and sometimes have additional syntax.

The syntax supported by SNMP is limited, and so the use of the textual convention mechanism for Standard MIB specification is growing. For similar reasons, the Passport Enterprise MIB also uses this mechanism to ease translation of the Passport component model onto the SNMP SMI, and to provide both increased user-friendliness and machine-processability.

Standard textual conventions

The Passport Enterprise MIB supports the use of standard textual conventions: both in the context of the specific Standard MIBs in which they are used and within Enterprise MIBs to allow Passport types to be mapped onto standard conventions where appropriate.

The table, “Support for SNMP standard textual conventions” (page 230), shows the standard textual conventions supported and the Passport base type used to encode them. Other textual conventions will be supported in future as required.

Table 45
Support for SNMP standard textual conventions

Textual convention	Passport base type
TAddress	String (Hex)
StorageType	enumeration
Evolution of the Interfaces Group of MIB II (RFC 1573)	
IANAifType	enumeration
OwnerString	String (0..255, ASCII)
InterfaceIndex	Decimal (1..2147483647)
Management Information Base for Frame Relay DTEs (RFC 1315)	
DLCI	Decimal
OSPF Version 2 Management Information Base (RFC 1253)	
AreaID	IP Address
RouterID	IP Address
Metric	Decimal (1..’FFFF’h)
BigMetric	Decimal (1..’FFFFFF’h)
Status	(enabled = 1, disabled = 2)
Validation	(valid = 1, invalid = 2)
PositiveInteger	Decimal (1..’FFFFFFFF’h)
HelloRange	Decimal (1..’FFFF’h)
UpToMaxAge	Decimal (1..3600)
DesignatedRouterPriority	Decimal (0..’FF’h)
TOSType	Decimal (0..31)

(Sheet 1 of 2)

Table 45 (continued)
Support for SNMP standard textual conventions

Textual convention	Passport base type
SNMP MIB Extension for the X.25 Packet Layer (RFC 1382)	
X121Address	String (0..17, ASCII)
(Sheet 2 of 2)	

Passport-specific textual conventions

Passport Enterprise MIB-specific textual conventions are formally defined in the MIB Module: *Nortel-MsCarrier-MscPassport-TextualConventionsMIB.*

Appendix D
Relationship between SNMP and OSI states

The SNMP agent assumes various OSI states indicative of whether or not the agent can process traps and requests. Each *ifTable* interface on a node has a SNMP status and an OSI state. SNMP status includes two status attributes:

- The *adminStatus* reflects the desired state. It can assume the values up, down, or testing.
- The *operStatus* reflects the current state. It can assume the values up, down, or testing.

The OSI state includes three state attributes:

- The *adminState* reflects the administration state of the component. It can assume the values locked, or unlocked.
- The *operState* reflects whether or not the component is operational. It can assume the value enabled, or disabled.
- The *usageState* omitted from the OSI state for interfaces because this status conveys no relevant information.

The table, “Interface states” (page 234), shows the valid combinations of SNMP status and OSI states. The table, “Interface state descriptions” (page 234), provides the descriptions of the *ifTable*’s status attributes (SNMP status) and state attributes (OSI state).

Table 46
Interface states

	SNMP status		OSI state event	
Interface state #	**adminStatus**	**operStatus**	**adminState**	**operState**
0	up	up	unlocked	enabled
1	up	down	unlocked	disabled
2	up	up	shuttingdown	enabled
3	up	down	locked	enabled
4	up	down	locked	disabled
5	test	test	locked	enabled
6	test	test	locked	disabled
7	down	down	locked	enabled
8	down	down	locked	disabled

Table 47
Interface state descriptions

Interface state #	Description
0	The normal up state.
1	Administration says the interface is up, however, operationally the interface is broken.
2	Locking the interface. Waiting for all users to quit.
3	Transiently locked (unlocks upon reboot).
4	Transiently locked and the interface is broken.
5	In test mode, the interface is healthy.
6	In test mode, the interface is broken.

(Sheet 1 of 2)

Table 47 (continued)
Interface state descriptions

Interface state #	Description
7	Administration says the interface is down, however, operationally the interface is “healthy”.
8	Administration says the interface is down and operationally the interface is “broken”.
(Sheet 2 of 2)	

Interface state change events

Events that cause the interface states to change are CAS events, SNMP events, and Interface events.

CAS events

The following are CAS events:

- *unlock* sets the interface’s OSI *adminState* to unlocked.
- *lock* sets the interface’s OSI *adminState* to shuttingDown.
- *lock -f* sets the interface’s OSI *adminState* to locked.

The table, “Interface state transitions based on CAS events” (page 235), shows the CAS events that cause the interface states to change.

Table 48
Interface state transitions based on CAS events

	Event -> new interface state		
Current interface state #	**unlock**	**lock**	**lock -f**
0	N/A	2	3
1	N/A	4	4
2	0	N/A	3
3	0	N/A	N/A
4	1	N/A	N/A
(Sheet 1 of 2)			

Table 48 (continued)
Interface state transitions based on CAS events

	Event -> new interface state		
Current interface state #	**unlock**	**lock**	**lock -f**
5	error	N/A	N/A
6	error	N/A	N/A
7	error	N/A	N/A
8	error	N/A	N/A

(Sheet 2 of 2)

SNMP events

The following are SNMP events:

- *up* sets the SNMP *adminStatus* to up.
- *test* sets the SNMP *adminStatus* to testing.
- *down* sets the SNMP *adminStatus* to down.

The table, “Interface state transitions based on SNMP events” (page 236), shows the SNMP events that cause the interface states to change:

Table 49
Interface state transitions based on SNMP events

	Event -> new interface state		
Current interface state #	**up**	**test**	**down**
0	N/A	5	7
1	N/A	6	8
2	N/A	5	7
3	N/A	5	7
4	N/A	6	8
5	0	N/A	7
6	1	N/A	8

(Sheet 1 of 2)

Table 49 (continued)
Interface state transitions based on SNMP events

	Event -> new interface state		
Current interface state #	**up**	**test**	**down**
7	0	5	N/A
8	1	6	N/A
(Sheet 2 of 2)			

Interface events

The following are interface events:

- *enable* heals the interface.
- *disable* breaks the interface.
- *reboot* restarts the node or interface.

The table, “Interface state transitions based on Interface events” (page 237), shows the interface events that cause the interface states to change:

Table 50
Interface state transitions based on Interface events

	Event -> new interface state		
Current interface state #	**enable**	**disable**	**reboot**
0	N/A	1	N/A
1	0	N/A	N/A
2	N/A	4	0
3	N/A	4	0
4	3	N/A	1
5	N/A	6	N/A
6	5	N/A	N/A
7	N/A	8	N/A
8	7	N/A	N/A

Appendix E Troubleshooting

This appendix provides suggestions for troubleshooting situations that may arise when an SNMP manager is trying to communicate with the SNMP agent on a Passport node. It is assumed that you have access to a text interface for the Passport system. The following topics are discussed:

Note: The IP connectivity to the node should be verified before using these procedures. See 241-5701-810 *Passport 7400, 15000, 20000 Configuring IP* for more information on specific problems related to UDP/IP connectivity.

No response from the SNMP agent

The most common cause of no response from the SNMP agent is that the SNMP component is either disabled or the provisioning is incorrect or incomplete.

If you receive no response from the SNMP agent:

1. Ensure that the *Snmp* component exists under the Virtual Router.
2. Ensure that the Virtual Router's OSI State is Unlocked, Enabled, Active.
3. Ensure that the *Snmp* components's OSI State is Unlocked, Enabled, Active.
4. Ensure that the *community* and *manager* components are set up with proper transport addresses and community strings.
5. Ensure that at least one community has been defined under the *Snmp* component for all originating transport addresses which use the community.
6. Ensure that the community and manager components are setup with proper transport addresses and community strings:
   - **a.** The community name in the SNMP message must match the *string* attribute for a *community* component using that transport domain.
   - **b.** If the *community* identified by the community name has *manager* subcomponents, the originating transport address must match that of one of the *manager* subcomponents.
7. Display the SNMP *MibIIStats* operational subcomponents and watch for changes in the statistics.
   - **a.** *inBadCommunityErrs* -- If they are increasing, the SNMP agent is receiving messages which are either badly formatted or corrupted. Check the SNMP manager.
   - **b.** *inAsnParseErrs* -- If they are increasing, the SNMP agent is receiving messages which are either badly formatted or corrupted. Check the SNMP manager.
   - **c.** *inBadVersion* -- If they are increasing, the SNMP agent is receiving messages with an SNMP version number other than 1. Check the configuration of the SNMP manager.
   - **d.** inPackets -- If they do not increment, the SNMP agent is not receiving the messages. Ensure that the IP address of the agent can be pinged. Ensure the SNMP component is Unlocked, Enabled, Active. check the installation and configuration of the SNMP manager.
8. Check to see if authentication failure traps are being generated as the SNMP messages are sent to the agent.

Generic error response to a set request

You can refer to the table “Summary of generic errors to a set request” (page 241) to trouble shoot generic error responses to a *set* request.

Table 51
Summary of generic errors to a set request

Item to verify	Steps required to troubleshoot
Before processing a set request from a SNMP manager, ensure no one is in provisioning mode.	• From a text interface, issue the command: **`display -o Prov`** • Examine the provisioningSession and/or provisioningUser attributes to see if someone is in provisioning mode. • If someone is in provisioning mode and cannot be contacted, you can force them out of provisioning mode by issuing the command: **`start -force Prov`**
Ensure the current and edit views are identical.l	• Issue the command: **`start Prov`** The response should include the statement: `The edit view is identical to the current view.` • If you do not get a confirmation in the response, and you want to make the views the same, issue the command: **`copy Prov`** ***Note:*** You will lose any prior changes made by a telnet user when issuing this command.

(Sheet 1 of 2)

Table 51 (continued)
Summary of generic errors to a set request

Item to verify	Steps required to troubleshoot
Ensure that the semantic check will not be a full semantic check.	• Issue the command: **`display Prov View/current`** If the *checkstate* attribute is something other than full, then a complete semantic check is required. • Start, check, and end the provisioning session: **`start Prov`** **`check Prov`** **`end Prov`**
Checking for alarms.	• Check to see if any alarms were generated around the same time that the *set* request was issued.
If you are still unable to resolve the problem, contact your Nortel Networks technical support representative.	
(Sheet 2 of 2)	

Note: By turning the spooling option on for the DCS log stream, you can access more detailed error responses from the SNMP log records, making troubleshooting that much easier. For more information on detailed error responses, see “Detailed error response to a set request” (page 246). For information on SNMP log records, see “Logging of SNMP Set Requests” (page 205).

SNMP set requests are failing

A SNMP *set* request may be completely rejected for the reasons indicated in the table “Reasons for complete rejection of set requests” (page 243). These errors are processed in the order shown.

Table 52
Reasons for complete rejection of set requests

Reason	Error (SNMP V1)	Error (SNMP V2)
The edit view is open.	genError	resourceUnavailable
The file system is currently locked.	genError	resourceUnavailable
Another provisioning command is in progress (for example, semantic checking) or fails (for example, the disks are not synchronized).	genError	resourceUnavailable
Another provisioning session is currently active.	genError	resourceUnavailable
The edit view is not the same as the current view.	genError	genError
The next semantic check should be a full semantic check.	genError	genError
Access to a variable is denied.	noSuchName	noAccess
A variable is not a leaf node (that is, it is not really a variable).	noSuchName	noAccess
A variable does not have write access.	noSuchName	noAccess
A value of the wrong type is supplied. For example, setting an INTEGER variable to a value of type OCTET STRING.	badValue	wrongType

(Sheet 1 of 3)

Table 52 (continued)
Reasons for complete rejection of set requests

Reason	Error (SNMP V1)	Error (SNMP V2)
A value of the wrong length is supplied. For example, setting a variable of type OCTET STRING (SIZE 6) to 'AB02'H.	badValue	wrongLength
An illegal value has been supplied. For example: • a *RowStatus* variable is set to *createAndWait* in the Enterprise MIB • an illegal INTEGER enumeration • an INTEGER value out of range • a string containing illegal characters	badValue	wrongValue
This *RowStatus* variable cannot be created because an invalid name has been supplied.	noSuchName	noAccess
This variable has not yet been created. This error applies to all variables except *RowStatus* variables.	noSuchName	noAccess
A *RowStatus* variable is set to createAndGo but not enough information has been supplied to create the row.	badValue	wrongValue

(Sheet 2 of 3)

Table 52 (continued)
Reasons for complete rejection of set requests

Reason	Error (SNMP V1)	Error (SNMP V2)
There are variables other than those specified for invoking the commit command.	genError	inconsistentValue
There are variables other than those specified for invoking the lock command.	genError	inconsistentValue
There are variables other than those specified for invoking the unlock command.	genError	inconsistentValue
A value is supplied that would normally be allowed but is currently inconsistent with values of other related variables.	genError	genError
A semantic check of an indicated modification has failed.	genError	genError
It was impossible to undo an indicated modification. (This only happens when operational information is modified.)	genError	genError
The response exceeds the maximum PDU size allowed: the response is too big for the manager.	tooBig	wrongLength

(Sheet 3 of 3)

If all SNMP requests are failing, perform the following steps:

1. Check the *community* has an access mode of readwrite.
2. Check that the *manager* has set privileges enabled.

3 Check that the *views* assigned to the *community* do not exclude the variables to be set.

Detailed error response to a set request

The response field in the SNMP set log record includes detailed responses for failed set requests. These responses can be captured for semantic checks. Semantic checks are invoked as part of the SNMP set request processing. Detailed error responses make it easier for you to determine the cause of SNMP set errors on Passport.

Responses for logged SNMP set requests are provided as follows:

- rich error responses provided by CAS are captured. If a set request fails due to an invalid set of Passport Enterprise MIB variables, the response field of the SNMP log record will contain a detailed error message to indicate the source of failure.
- error responses are provided for SNMP set requests failing due to an error with setting a standard MIB variable
- error responses are provided for SNMP agent errors related to an SNMP set request

For example, if you try to add a *logicalProcessor* component without setting its *mainCard* attribute, you will get an error. The SNMP manager will only get an error status of “bad value” but the log record will display the following detailed response of “The *mainCard* attribute is not defined”.

If you choose to use this feature, spooling for the DCS log stream must be turned on. See table “Possible error responses to SNMP set requests” (page 247) to realize the extent of the detail this feature has to offer. These responses can be compared to those in table “Reasons for complete rejection of set requests” (page 243).

Table 53
Possible error responses to SNMP set requests

Error response	Description
Resource unavailable: Unable to start the provisioning session.	An SNMP set request produces this response when the SNMP Agent tries to start provisioning but another session is already in provisioning mode or the current view is not the same as the edit view.
Resource unavailable: SNMP has been forced out of provisioning mode.	An SNMP set request produces this response when the SNMP Agent has been forced out of provisioning mode.
Resource unavailable: The switch requires a full semantic check.	An SNMP set request produces this response when the set request cannot be processed because the switch requires a full semantic check.
An unknown community was used for the set request.	An SNMP set request produces this response when the SNMP set request uses an unknown community.
The SNMP manager does not belong to the specified SNMP community.	An SNMP set request produces this response when the SNMP set request is from an SNMP Manager that is not provisioned under the current community.
The SNMP Manager belongs to a read only community.	An SNMP set request produces this response when the SNMP set request uses a community that has read-only access.
The SNMP manager does not have set privilege.	An SNMP set request produces this response when the SNMP set request is from an SNMP Manager that does not have set privileges.
The response PDU is too big. The limit is 1400 octets.	An SNMP set request produces this response when the size of the Response PDU exceeds the maximum PDU size which is 1400 octets.
An invalid object identifier found in VarBind/n	An SNMP set request produces this response when the SNMP set request has an invalid object identifier.

(Sheet 1 of 2)

Table 53 (continued)
Possible error responses to SNMP set requests

Error response	Description
The object identifier specified in VarBind/n is excluded by the view.	An SNMP set request produces this response when the SNMP set request has an object identifier that is excluded by the SNMP view.
The object identifier specified in VarBind/n is not a leaf node.	An SNMP set request produces this response when the SNMP set request has an object identifier that is not a leaf node.
The object identifier specified in VarBind/n is not writable.	An SNMP set request produces this response when the SNMP set request has an object identifier that is not writable.
The value specified in VarBind/n has the wrong type.	An SNMP set request produces this response when the SNMP set request has an object identifier with a wrong type.
Set failed on Component/n with the error status:/m	An SNMP set request produces this response when the SNMP set request failed on adding a component.
Set failed on Attribute/n, m with the error status:/o	An SNMP set request produces this response when the SNMP set request failed on setting an attribute.
Set failed on VarBind/n with the error status:/m	An SNMP set request produces this response when the SNMP set request failed on a Standard MIB variable.
Set failed because it would cause CP reboot.	An SNMP set request produces this response when the SNMP set request will cause a CP reboot, which is not supported by SNMP.

(Sheet 2 of 2)

Time-out on SNMP set requests

Under typical network conditions, a SNMP set request should not fail due to a time out. However, given the fact that SNMP uses UDP (User Data Protocol), time outs can still occur.

When performing a set request on the *mscProvCommitFileOption* variable, a time out can occur because:

- the SNMP set request PDU is lost
- the time out value on the SNMP request is less than five seconds versus the recommended time out value of 10-20 seconds

Any time out occurring from the perspective of the SNMP manager implies that no response was received. If a time out does occur, SNMP managers are configured to resend the packet.

Before a SNMP set request is resent, you should determine if the SNMP set request was lost. To confirm this, refer to the following sections:

- for a commit command, refer to “Confirming the status of the configuration file” (page 208)
- for a lock command, refer to “Confirming that the FrUni Dlci link is locked” (page 210)
- for an unlock command, refer to “Confirming that the FrUni Dlci link is unlocked” (page 211)

SNMP get requests are failing

A SNMP *get* request may be rejected completely. The reasons and associated errors are shown in the table “Reasons for complete rejection of SNMPv1 get request” (page 249). For the reasons and associated errors on particular variables, see the table “Reasons for SNMPv1 get errors on particular variables” (page 250).

Table 54
Reasons for complete rejection of SNMPv1 get request

Reason	Error
A requested variable is write-only.	genError
(Sheet 1 of 2)	

Table 54 (continued)
Reasons for complete rejection of SNMPv1 get request (continued)

Reason	Error
The request exceeds the maximum PDU size allowed: the request is too big for the Agent.	tooBig
The response exceeds the maximum PDU size allowed: the response is too big for the Manager.	tooBig
(Sheet 2 of 2)	

Table 55
Reasons for SNMPv1 get errors on particular variables

Reason	Error
Access to this variable is denied.	noSuchName
There is no such variable. It is impossible that the variable could exist according to the MIB Modules supported by the Agent.	noSuchName
The variable is valid and could exist, but does not at the moment. This exception occurs when variables from a non-existent row are requested.	noSuchName

SNMP getNext requests are failing

Table 56
Reasons for complete rejection of SNMP getNext requests

Reason	Error
The request exceeds the maximum PDU size allowed: the request is too big for the agent.	tooBig
The response exceeds the maximum PDU size allowed: the response is too big for the manager.	tooBig

An *endofMIB* indication is returned for variables at the end of the MIB.

Software errors

Should a software error occur while processing a SNMP request, a software alarm is generated and a *genErr* error is returned. This may occur for *get*, *getNext*, or *set* requests.

Traps are not being received

You should ensure that the event is configured to receive traps before you troubleshoot. See “Passport alarms as traps” (page 132). Perform the following steps to troubleshoot the problems associated with traps not being received:

1. Ensure that the *Snmp* component is Unlocked, Enabled, and Active.
2. Ensure that the *manager* has trap privileges enabled.
3. If the trap is an authentication trap, ensure that the generation of authentication failure traps is enabled by checking the *Snmp* component’s *EnableAuthenTraps* attributes.
4. Check that the data collection system’s Trap agent queue size is provisioned to a suitable value.

   ```
   display Lp/0 Eng Ds/trap
   ```

5. Monitor the DCS Trap Collector and agent operational statistics, the *Snmp* component’s *trapsProcessed* and *trapsDiscarded* attributes.

Authentication failure traps received

A trap is generated whenever there is an authentication failure if *EnableAuthenTraps* is enabled. However, a manager receiving these traps may not be the one causing the authentication failures. To troubleshoot, perform the following steps:

1. Poll the operational attributes, *lastAuthFailure* and *mgrOfLastAuthFailure* under the *Snmp* component.
2. From these operational attributes, determine the time of the last authentication failure and the IP address of the manager causing the last authentication failure.
3. Ensure that the *community* and *manager* components are set up with the proper community strings.

Passport 7400, 15000, 20000
SNMP Guide

Release 5.2

Publication: 241-5701-300
Document status: Standard
Document version: 5.2S1
Document date: November 2003
Printed in Canada

NORTEL NETWORKS