NORTEL NETWORKS

Preside Multiservice Data Manager

DPN SNMP Agent

User Guide

241-6001-210

Preside Multiservice Data Manager

DPN SNMP Agent

User Guide

Publication: 241-6001-210
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003

Printed in Canada

Publication history

December 2003

14.3RSUP Standard
Commercial availability

Contents

About this document

The DPN SNMP Agent (hereafter referred to as the *DPN Agent*) is the implementation of an SNMP proxy agent for the Nortel Networks DPN-100 switching equipment. The *DPN Agent* integrates DPN equipment surveillance into any third-party network management system supporting the SNMP protocol. This document assumes the *DPN Agent* has been installed along with Preside Multiservice Data Manager (MDM).

The following topics are discussed in this section:

- "Who should read this document and why" (page 11)
- "What you need to know" (page 11)
- "How this document is organized" (page 12)
- "What's new in this document" (page 13)
- "Text conventions" (page 13)
- "Related documents" (page 14)

Who should read this document and why

This document is intended for those involved in the deployment and operation of the *DPN Agent* with a production DPN network.

What you need to know

To use the *DPN Agent*, you must know how to log on to the Preside MDM workstation and how to work with the Preside Multiservice Data Manager (MDM) workstation user interface. A good working knowledge of the network model and the RFC documents listed in "Related documents" (page 14) is also required. You must also be familiar with the UNIX operating system. If you are unfamiliar with any of these, you should read about them in the references listed in this section before attempting the operations detailed in this guide.

How this document is organized

The information in this guide is organized as follows:

"Overview" (page 15) describes the functionality and provides a high-level overview of the *DPN Agent*.

"Engineering the workstation for the DPN Agent" (page 19) describes the engineering requirements of the workstation host for the *DPN Agent*.

"Installing the DPN Agent" (page 21) describes the installation procedures required for the *DPN Agent* on a Preside Multiservice Data (MDM) workstation. Also included is a procedure for the deinstallation of the *DPN Agent*.

"Administration and configuration" (page 27) describes the configuration and administration available to engineer the *DPN Agent* to run effectively.

"Operations" (page 41) details how the *DPN Agent* is invoked.

"DPN Enterprise MIB" (page 71) provides a description and specification of the DPN Enterprise MIB.

"Detailed Port MIB Mappings" (page 113) describes the detailed mappings from the information obtained from the DPN NCS to the appropriate *DPN Agent* SNMP MIB variables.

"Engineering measurements" (page 131) lists the X.25 traffic generated by the *DPN Agent* and the CPU utilization within the PE running the OA.

"Event configuration" (page 149) describes how to configure DPN Agent network events. The *DPN Agent* can be used with any SNMP network management application and we have chosen HP OpenView for example purposes, only.

What’s new in this document

There are no changes in this document for this release.

Text conventions

This document uses the following text conventions:

- `nonproportional spaced plain type`

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

- **`nonproportional spaced bold type`**

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

- *italics*

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

  Words that appear in italics in text are for naming.

- `[optional_parameter]`

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

- `<general_term>`

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

- UPPERCASE,lowercase

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

- |

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

- ...

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

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

Related documents

See the following documents for related information:

- Internet Standard RFC 1155, *Structure and Identification of Management Information for TCP/IP-based Internets*.
- Internet Standard RFC 1157, *A Simple Network Management Protocol*.
- Internet Standard RFC 1212, *Concise MIB Definitions*.
- Internet Standard RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB II*.
- 241-1001-303 *DPN-100 Operator Commands and Responses*, Parts 1 & 2.
- 241-2001-102 *DPN-100 Network Control System User Guide*.
- 241-6001-100 *Preside MDM Installer Guide*.
- 241-6001-303 *Preside MDM Administrator Guide*
- 241-1001-506 *DPN-100 Alarm Console Indications*

Chapter 1 Overview

This chapter provides you with an understanding of the following functions and features of the DPN SNMP Agent:

- “About the DPN SNMP Agent” (page 15)
- “Basics” (page 16)

About the DPN SNMP Agent

The *DPN Agent* acts as a *proxy* for DPN switches by interfacing with the network equipment using existing DPN-specific protocols and presenting this management information to manager platforms by means of SNMP.

The *DPN Agent* executes on a SUN workstation and communicates with the DPN Network through the communications facilities provided by the Preside Multiservice Data Manager (MDM).

Activities you can perform with the DPN Agent

The *DPN Agent* provides you with the means to perform the following activities with your SNMP manager software application:

- retrieve management information from DPN network devices
- modify a subset of the management information
- receive unsolicited DPN network alarms and logs in the form of SNMP traps
- perform alarm clearing of active alarms within the DPN network.

Basics

The *DPN Agent* supports automatic network discovery of UTP (Network Links or Trunks), LAPB/X.25, frame relay, Interactive Terminal Interface (ITI), Token-Ring, HDLC Transparent Data Service (HTDS), SNA XPAD, and LAPD/X.25 ports. The RFC 1213 *ifTable* and DPN Enterprise *dpnIfTable* MIB variables are populated with the network discovery data.

The *DPN Agent* is designed to be independent of the DPN-100 network generic software. It interworks with all of the Nortel Networks-supported software releases.

The *DPN Agent* queries the DPN network and receives responses by issuing AM and RM operator commands and receiving the responses. These commands are issued by connecting to the Network Control System (NCS) Operation Agents used to control and route the management information and operator commands within a DPN network.

Capabilities

The *DPN Agent* supports a subset of the variables outlined in the MIB II definitions; the *system* group, *snmp* group and the *ifTable*. The statistics variables of the *ifTable* rows (for example, *ifInOctets*) have been removed from the *ifTable* support and moved into the DPN Enterprise MIB.

In addition to the standard MIBs supported, the *DPN Agent* also implements the DPN Enterprise MIB whose details are documented in “DPN Enterprise MIB” (page 71). The DPN Enterprise MIB defines DPN-specific extensions for the control and collection of statistics variables (for example, *dpnIfInOctets*) on DPN network ports. The DPN Enterprise MIB also defines DPN alarm and log trap formats and an interface for the clearing of the alarms within the DPN network.

The *DPN Agent* supports the SNMP PDUs: GET, GET-NEXT, and SET (only for some *system* group variables) as outlined in RFC 1157. The *DPN Agent* communicates on default UDP port 161 on the host machine for the *DPN Agent*. Traps (for *DPN Agent* restart, and port up and down) are sent out from UDP port 162 to the IP addresses configured in the *DPN Agent* setup files.

The *DPN Agent* supports a maximum SNMP PDU size of 2048 bytes.

MDM Dependencies

The *DPN Agent* establishes inter-process communications (IPC) connections to the Preside Multiservice Data Manager (MDM) NCSCOM (NCS Communications Manager) and CCIF (Communications Channel Interface) servers in order to inject operator commands into the DPN network in the same fashion as the MDM Command Console application. See "Host MDM" (page 51) for reasons the *DPN Agent* requires MDM.

Chapter 2
Engineering the workstation for the DPN Agent

This chapter contains information required to engineer a workstation to run the *DPN Agent*. This includes the following topics:

- “Suggested configuration” (page 19)
- “Hardware requirements” (page 20)

Suggested configuration

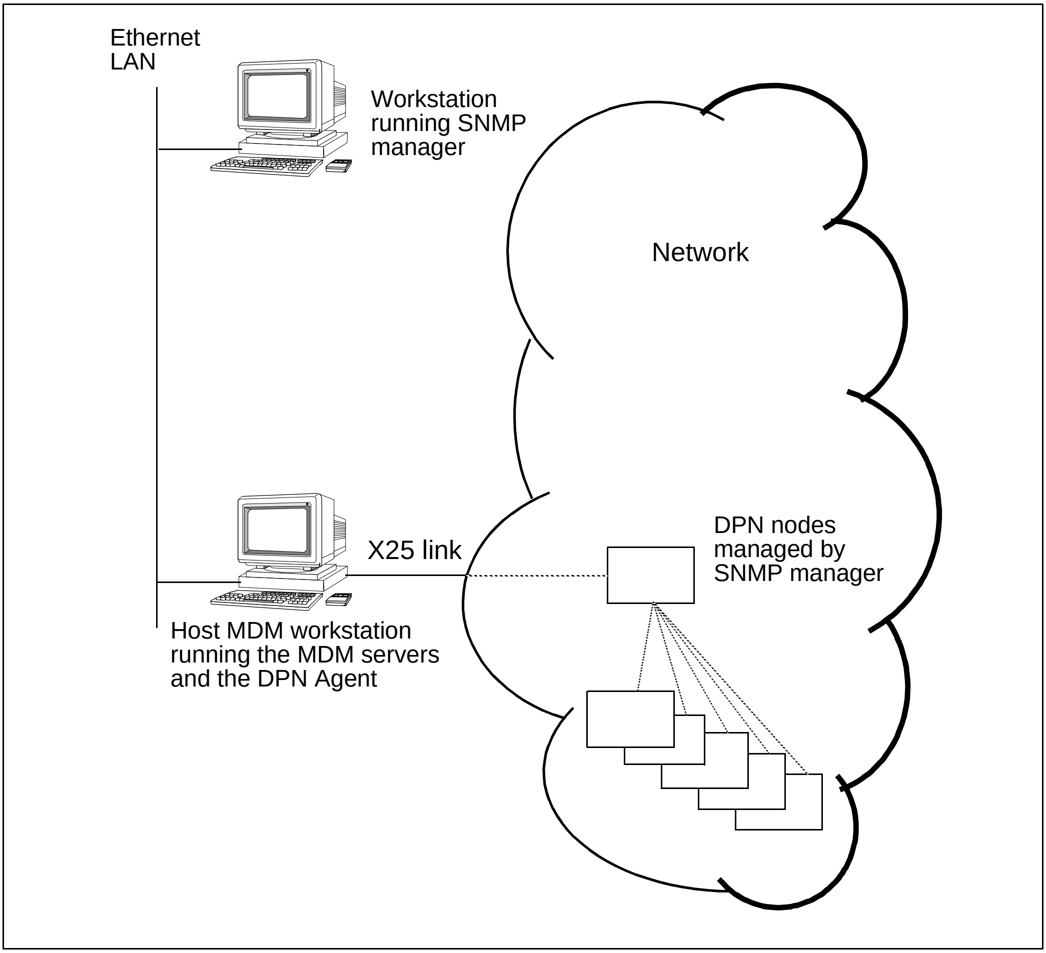
The suggested workstation configuration for the *DPN Agent* involves one workstation to run the *DPN Agent* software and to host Preside Multiservice Data Manager (MDM) for access and transport of management communications into the DPN network (mandatory)

The third-party SNMP manager platform can be deployed as an independent workstation. Alternatively MDM, the *DPN Agent*, and the SNMP manager can be deployed onto a single workstation.

The figure “Elements within a DPN Agent/MDM configuration” (page 20) shows the various elements within a *DPN Agent* configuration.

The workstation(s) running the SNMP manager application and the DPN Agent are Ethernet LAN connected to exchange management information using the SNMP protocol.

Figure 1
Elements within a DPN Agent/MDM configuration



Hardware requirements

See 241-6001-100 *Preside MDM Installer Guide* for a description of the hardware requirements necessary to host the Preside Multiservice Data Manager (MDM). Additional disk space and memory requirements may be necessary to accommodate the *DPN Agen*t software.

Chapter 3
Installing the DPN Agent

This chapter describes how to install the *DPN Agent* on an Preside Multiservice Data Manager (Preside MDM) host.

The following topics are discussed in this chapter:

- "Installing DPN Agent on a Preside MDM workstation" (page 21)
- "Installing a DPN Agent software license" (page 22)
- "Creating the distribution hierarchy" (page 24)
- "Migrating from previous releases" (page 24)

Also provided is a procedure to remove the *DPN Agent* from a workstation. See "De installing the DPN Agent" (page 25).

Installing DPN Agent on a Preside MDM workstation

The following procedure is intended to guide you on the over-all process of installing the *DPN Agent* software on a Preside MDM workstation.

Setting up the DPN Agent

1 See 241-6001-100 *Preside MDM Installer Guide* for procedures to install and configure Preside MDM. The Preside MDM software package contains the *DPN Agent* software.

2 Install the *DPN Agent* license. See "Installing a DPN Agent software license" (page 22).

3 Run the *DPN Agent* installation script. This script creates and populates the configuration directories required by the *DPN Agent*. See "Creating the distribution hierarchy" (page 24).

Installing a DPN Agent software license

Licence information is provided to you along with the software on compact disk. If you do not have this information, contact your Nortel Networks Customer Representative.

The license information consists of a license key and a customer identifier. Installing an license involves entering the license information into:

- the ***license key*** file:
  /etc/opt/Magellan/LIClicences.cfg
- the ***customer identifier*** file*:*
  /etc/opt/Magellan/LICcustName.cfg

The utility *config-mdp-licenses* is provided for adding license information into these files.

Licence key file

The license key file contains one or more license keys. A license key is a single line of information with a number of fields. You must enter field information exactly as provided, otherwise the software does not run properly.

```
<prod_name> <release> <custid> <hostid> <start_date>
<expiry_date> <options> <password>
```

where:

`<prod_name>`

is a three-character code that represents the product.

`<release>`

is a software release identifier with up to 6 characters (for example, ***R10.6****).*

`<custid>`

is a customer identifier consisting of up to 10 characters, with no spaces. For example, *BellCust.* This customer identifier must match the entry in the customer identifier file /etc/opt/Magellan/LICcustName.cfg.

`<hostid>`

is the host name of the workstation or server on which the software is to run, consisting of up to 8 characters. The default is *ANY*.

`<start_date>`

is the date at which the license becomes valid, consisting of 8 characters in the form *<year><month><day>*. For example: 19970417.

`<expiry_date>`

is the date at which the license expires, consisting of 8 characters in the form *<year><month><day>*. For example: 19980417.

`<options>`

is an options code, consisting of up to 8 hexadecimal characters that translates into a bitmap which identifies the software options (software subsets) that the license key entitles you to run.

`<password>`

is an 11 hexadecimal character encrypted password that the licensing software uses to determine if the information entered into the other non-encrypted fields is valid.

Customer identifier file

The customer identifier file contains a single customer identifier. The customer identifier entered in this file must match the customer identifier entered in field *custid* in the license key file.

Creating the distribution hierarchy

To create the *DPN Agent* distribution hierarchy of directories and to populate these directories with the necessary files, you must execute the installation script *installdsp*.

See "Migrating from previous releases" (page 24) if the *DPN Agent* is already configured on the current workstation.

Running the DPN Agent installation script

1 Change to the directory you want to contain the *DPN Agent* configuration. For example:

```
cd /opt/MagellanNMS
```

2 Run the *DPN Agent* installation script by typing

```
/opt/MagellanNMS/bin/installdsp
```

The installation script creates the *DPN Agent* directories *dpnAgent/bin, dpnAgent/cfg, dpnAgent/doc and dpnAgent/log* as subdirectories of the current directory.

Migrating from previous releases

The process of migrating from a previous release of *DPN Agent* is the same as the process for the original installation. You must re-install Preside MDM and execute the *DPN Agent* installation script *installdsp*. See "Installing DPN Agent on a Preside MDM workstation" (page 21).

The installation script *installdsp* performs the following:

- the *dpnProxy* and *dpnPoller* executables are replaced
- the *DPN-100.mib* file is replaced
- the *dpnProxy.map* and *dpnProxy.ver* files are replaced

- files *dpnProxy.cnf*, *dpnProxy.oas*, and *dpnProxy.sel* are examined for changes. If you have modified any of these configuration files, the *installdsp* script does not overwrite them.

  To use a new *DPN Agent* feature requiring changes to any of these files, you must manually edit the existing version of the configuration file. If your reason for migrating to the latest DPN Agent release does not involve additions to the above configuration files (for example, a new service), you do not need to make any configuration file changes.

  For example, to use the *DPN Agent* for a new supported service, you must add the service name, as identified by either the Release Supplement or this document, to the *dpnProxy.sel* file. See “dpnProxy.sel” (page 36) for more information.

  For example, to use the *mapAlarmCode* option of the *dpnProxy.cnf* file, you may need to add the *mapAlarmCode* line to the *dpnProxy.cnf* file. See “dpnProxy.cnf” on page 27 for more information.

De installing the DPN Agent

You only de-install the *DPN Agent* if you need to completely remove the application from a workstation. If you are installing a new version of *DPN Agent*, the installation script *installdsp* replaces existing copies of *DPN Agent* software and configuration files. For more information about replacing existing *DPN Agents*, see “Migrating from previous releases” (page 24).

If more than one *DPN Agent* is installed on the same workstation, de installing one *DPN Agent* does not de install the others. *DPN Agent* instances exist in separate directory locations. Ensure that you terminate the correct instance of the *DPN Agent* server using the unique SVM registered name associated with server.

De installing a DPN Agent

1 Terminate the executing *DPN Agent*. See “Terminating the DPN Agent” (page 64).

2 Login to the platform as userID *root*.

3 Change directory to the location of the *DPN Agent* software. For example, type the following command at the UNIX command line:

```
cd /opt/MagellanNMS/dpnAgent
```

4 To remove all *DPN Agent* software and configuration files, type the following command at the UNIX command line:

```
rm -R *
```

5 Change directories to the parent directory. For example, type the following command at the UNIX command line:

```
cd ..
```

6 Remove the *DPN Agent* parent directory. For example, type the following command at the UNIX command line:

```
rmdir dpnAgent
```

You have completed the steps necessary to remove the *DPN Agent* from a workstation.

Chapter 4
Administration and configuration

This chapter provides a description of the various configuration files used to define and control the operation of the *DPN Agent*. See “Configuration files” (page 27).

This chapter assumes that you have installed or have access to a workstation running release R10.9, or later, of Preside MDM.

Configuration files

The *DPN Agent* makes use of five configuration files located in */opt/dpnAgent/cfg*, each of which you must configure for the *DPN Agent* to correctly operate within a production environment. These configuration files are:

- “dpnProxy.cnf” (page 27)
- “dpnProxy.oas” (page 32)
- “dpnProxy.sel” (page 36)
- “dpnProxy.map” (page 37)
- “dpnProxy.ver” (page 39)

dpnProxy.cnf

This configuration file defines the operational characteristics of the *DPN Agent*. It is the file that you are most likely to modify. The figure “Sample dpnProxy.cnf” (page 28) shows a sample *dpnProxy.cnf* file.

Unknown configuration options are ignored and logged by the *DPN Agent* during file processing.

> ***Note:*** The maximum size of each field in file *dpnProxy.cnf* is 80 characters.

Figure 2
Sample dpnProxy.cnf

```
######################################################################
#                                                                    #
# File:         dpnProxy.cnf                                         #
#                                                                    #
# Description:  This file defines the operational characteristics    #
#               of the DPN SNMP Agent.                               #
#                                                                    #
# Format:       Each line consists of a tuple containing             #
#               two fields:                                          #
#                                                                    #
#               (1) configuration option which specifies the option  #
#                   being configured; and                            #
#               (2) option value consisting of one or more           #
#                   whitespace seperated strings                     #
#                                                                    #
#               Option and value are seperated by whitespace.        #
#                                                                    #
#               Case and whitespace is ignored.                      #
#               Comment lines begin with #.                          #
#                                                                    #
######################################################################

sysDescr            Magellan DPN SNMP Agent
sysContact          Somebody
sysLocation         Somewhere
readCommunity       public standard
readCommunity       cust1 dpn
writeCommunity      cust2 dpn
trapDest            mickey public HP_OpenView
mapAlarmCode       TRUE
```

Various configuration options and their associated values are defined in the example file. The various options within this configuration file along with their associated description and possible values are found in the table “dpnProxy.cnf Configuration Options” (page 29).

Table 1
dpnProxy.cnf Configuration Options

Option	Description
sysDescr	**sysDescr <descriptionString>** This option specifies the value of the *sysDescr* RFC 1213 MIB variable used by the *DPN Agent*. This option can specify any value which can consist of one or more string tokens separated by whitespace.
sysContact	**sysContact <contactString>** This option specifies the value of the *sysContact* RFC 1213 MIB variable used by the *DPN Agent*. This option can specify any value which can consist of one or more string tokens separated by whitespace.
sysLocation	**sysLocation <locationString>** This option specifies the value of the *sysLocation* RFC 1213 MIB variable used by the *DPN Agent*. This option can specify any value which can consist of one or more string tokens separated by whitespace.
readCommunity	**readCommunity <communityName> <mibView>** This option specifies a *DPN Agent* supported read-only community name and the associated MIB variables accessible through this community name. The *mibView* token can have the following values: *standard* which provides access to standard MIB definitions *dpn* which provides access to DPN enterprise MIB definitions *combined* which provides access to all MIB definitions A minimum of one read community must be defined for MIB read access. A maximum of 20 read-only communities can be specified using individual *readCommunity* entries.
(Sheet 1 of 3)	

Table 1 (continued)
dpnProxy.cnf Configuration Options

Option	Description
writeCommunity	**writeCommunity <communityName> <mibView>** This option specifies a *DPN Agent* supported read-write community name and the associated MIB variables accessible using this community name. The *mibView* token can have the following values: *standard* which provides access to standard MIB definitions *dpn* which provides access to DPN enterprise MIB definitions *combined* which provides access to all MIB definitions. A minimum of one write community must be defined for MIB write access. A maximum of 20 read-write communities can be specified using individual *writeCommunity* entries.
trapDest	**trapDest <host> <communityName> [<platformQualifier>]** This option specifies a destination where the *DPN Agent* is to forward generated SNMP traps. This option can specify a value consisting of two (2) or three (3) string tokens. The first token is the destination address of the receiving management platform (that is, either IP address or host name) The second token is the community name for traps to this destination to use. The third token is a management platform qualifier. Currently, the only management platform qualifier supported is *HP_OpenView* which directs the *DPN Agent* to include the OpenView specific trap severity override variable binding (1.3.6.1.4.1.11.2.17.2.5.0) as documented in the HP *trapd.conf* manual page within the generated traps. The absence of the management platform qualifier directs the *DPN Agent* to not perform any platform specific trap processing. A maximum of 20 trap destinations can be specified for the *DPN Agent* with each destination being defined in a separate *trapDest* entry.
(Sheet 2 of 3)	

Table 1 (continued)
dpnProxy.cnf Configuration Options

Option	Description
mapAlarmCode	**mapAlarmCode [TRUE | FALSE]** This option indicates how the *DPN Agent* generates DPN Enterprise traps. It specifies what decimal integer is used as the specific trap code as defined by RFC1157. Selecting TRUE results in SNMP V1 traps that are generated by DPN alarm or log events to have the specific trap code set to the decimal equivalent of the DPN alarm code associated with the DPN alarm or log. TRUE is the default. Selecting FALSE causes the specific trap code to be set to 1 for all traps generated as a result of a DPN alarm or log. DPN Enterprise alarm codes are displayed as hexadecimals, but SNMP object identifiers (oids) are a sequence of decimal integers. For example, DPN alarm code 200140FF is indicated as 1.3.6.1.4.1.562.2.8.5.0.0.536953087 with mapAlarmCode set to TRUE and is indicated as 1.3.6.1.4.1.562.2.8.5.0.0.1 with mapAlarmCode set to FALSE. 1.3.6.1.4.1.562.2.8.5.0.0.1 is a concatenation of the DPN Enterprise oid and a 0 followed by the actual trap identifier. HP OpenView uses these oids to identify events or traps. When HP-OV is configured it determines that two traps are defined, namely .1.3.6.1.4.1.562.2.8.5.0.1 and .1.3.6.1.4.1.562.2.8.5.0.2 and automatically adds the events. This requires the event oid to be modified with the final 1 being replaced with a wildcard, namely *. For management platforms which do not support wildcards the mapAlarmCode can be set to FALSE. See “Event configuration” (page 149) for a description of event configuration using HP OpenView.
(Sheet 3 of 3)	

User Communities

MIB access (through the *readCommunity* and *writeCommunity* configuration option) is only permitted for the defined communities. Any management request received by the *DPN Agent* on an unknown community name results in two actions: an authentication failure trap is sent to all configured management platforms, and the standard MIB variable *snmpInBadCommunityNames* is incremented.

A MIB view is also associated with each and every community name. This MIB view can be used to restrict MIB variable access by specifying one of the following values:

- *standard* indicates the community name can access only the standard MIBs supported by the *DPN Agent* (that is, RFC 1213 supported variables)
- *dpn* indicates the community name can access only the *DPN Agent* enterprise MIB (refer to in “DPN Enterprise MIB” (page 71))
- *combined* indicates the community name can access both the standard and enterprise MIBs of the *DPN Agent*

Read-only and read-write access is controlled by the *readCommunity* and *writeCommunity* configuration options respectively.

> ***Note:*** The MIB view construct does not support a finer level of granularity for MIB variable access. In particular, a DPN network may contain multi-customer ownership of ports in which management stations of any customer may access MIB variables associated with any port and, therefore, any customer. The MIB view construct is not meant as a replacement for customer network management (CNM) wherein MIB variables are only visible to owning SNMP managers. CNM is not supported by the *DPN Agent* at this time.

The community name associated with SNMP trap destinations simply specifies the community name to be inserted into the trap itself.

dpnProxy.oas

The *DPN Agent* communicates with the DPN Network Control System (NCS) operations agents (OAs) for network access and topology discovery. Associated with each and every OA is a login process required to establish a communications session. The *dpnProxy.oas* file defines the set of one or more OAs that the *DPN Agent* can directly communicate with and their required login parameters (that is, capability and password). The figure “Sample dpnProxy.oas” (page 33) shows a sample *dpnProxy.oas* file.

The following information requires a working knowledge of NCS and OAs and you should refer to 241-2001-102 *DPN-100 Network Control System User Guide* for details regarding NCS and OA functionality.

Figure 3
Sample dpnProxy.oas

```
######################################################################
#                                                                    #
# File:         dpnProxy.oas                                         #
#                                                                    #
# Description:  This file contains the list of network OAs and       #
#               associated capability and passwords for each OA that #
#               the DPN SNMP Agent may wish to communicate with.     #
#                                                                    #
#               Important:  The first OA listed in this table        #
#               =========   should be that of the top-level OA       #
#                           of the network.                          #
#                                                                    #
#                           The top-level OA will become the         #
#                           default OA for requests that cannot be   #
#                           dispatched to the lower-level OA for a   #
#                           specific module.                         #
#                                                                    #
#                           Network auto-discovery is rooted at      #
#                           the top-level OA.                        #
#                                                                    #
# Format:       Each OA tuple consists of the OA name, capability    #
#               and password entries.                                #
#                                                                    #
# Warning:      The OA Name must match the name specified in the NMS #
#               configuration file HGDS.cfg (OAMember) and further   #
#               this MUST be the name of the OA as defined in        #
#               service data                                         #
#                                                                    #
#               Case and whitespace is ignored.                      #
#               Comment lines begin with #.                          #
#                                                                    #
######################################################################

#   OA Name            Capability            Password
#   =======            ==========            ========
    MYOA               OPER1                 DUMBO
```

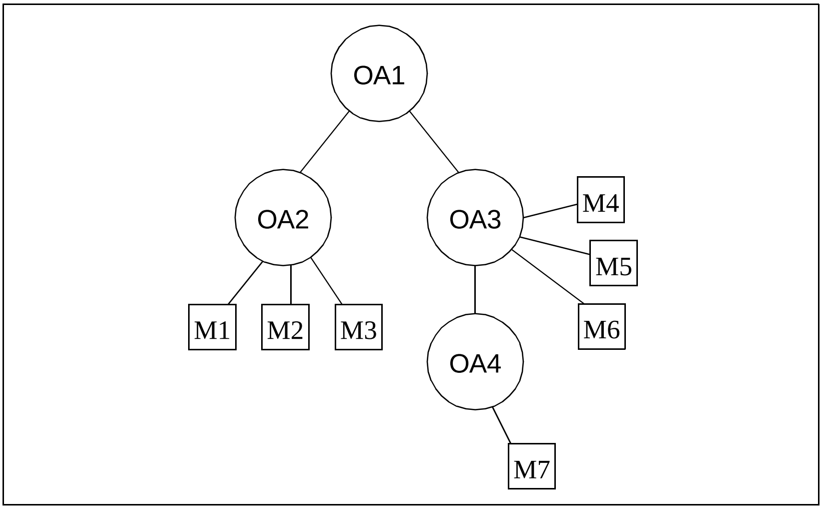
A minimum of one OA must be configured within the *dpnProxy.oas* configuration file to provide *DPN Agent* access into the DPN network.

Due to the possible hierarchical layering of a network’s OAs, the first OA configured within the *dpnProxy.oas* file is viewed as the coordinating or top-level OA of the hierarchy and is considered as the *root* of all other OA definitions. The top-level OA becomes the default root for command processing for those modules and ports whose controlling OA has not been defined within the *dpnProxy.oas* configuration file.

The top-level OA is also the point at which the *DPN Agent* accesses the DPN network alarm and log streams. As a result, the *DPN Agent* should be rooted at an OA to which all subordinate network devices forward their alarms.

See the figure “Partial network OA hierarchy” (page 34) for an example of deploying the *DPN Agent* in a partial network OA hierarchy.

Figure 4
Partial network OA hierarchy



By defining OA1 as the top-level OA, the *DPN Agent* can monitor ports within all seven modules (that is, M1 through M7) since all modules are subordinate to OA1. If OA3 is defined as the top-level OA and no other OAs are defined, then the *DPN Agent* can only support network elements subordinate to OA3 and OA4 - elements within OA1 and OA2 would not be visible.

> ***Note:*** *DPN Agent* network discovery (refer to “Port discovery mode” (page 55)) is limited to those network elements contained within the top-level OA and its subordinate OAs. Therefore, a *DPN Agent* rooted at OA3 running with auto-discovery enabled would only discover elements controlled by either OA3 or OA4.

The concept of OA hierarchies and rooting the *DPN Agent* to a particular OA permits a very flexible deployment capability, ranging from a single *DPN Agent* responsible for network-wide management to a localized *DPN Agent* responsible for exerting control over specific segments of a network. OA rooting combined with network discovery enables the *DPN Agent* to be deployed into any required management configuration.

NCS capability requirements

NCS capability requirements for OA access and the proper operation of the *DPN Agent* are as follows:

- If DPN-100 network logs are to be translated to SNMP traps and forwarded to configured management stations, the capability associated with the top-level OA must be configured within NCS to permit log data streams out of the network.
- If DPN-100 network alarms are to be translated to SNMP traps and forwarded to configured management stations, the capability associated with the top-level OA must be configured within NCS to permit alarm data streams out of the network.
- The impact associated with all OAs defined within *dpnProxy.oas* must be PASSIVE at the minimum.

Refer to the section “OA hierarchy probe” (page 51) for information on probing an OA hierarchy to determine if the above requirements are satisfied.

As a result of any failure to meet the above requirements, the *DPN Agent* cannot receive network alarms and logs or perform network topology discovery.

dpnProxy.sel

The *dpnProxy.sel* file defines the types of ports the *DPN Agent* is to support. During network topology discovery, ports not supported are discarded and are not populated in the *DPN Agent* MIB. The figure “Sample dpnProxy.sel” (page 36) shows a sample *dpnProxy.sel* file.

Figure 5
Sample dpnProxy.sel

```
######################################################################
#                                                                    #
# File:         dpnProxy.sel                                         #
#                                                                    #
# Description:  This file defines the types of ports the DPN SNMP    #
#               Agent will support.  Commenting out any of the       #
#               defined port types causes ports of that specific     #
#               type to be ignored during network topology           #
#               discovery and subsequently not modelled within       #
#               the DPN SNMP Agent MIB.                              #
#                                                                    #
# Format:       One port type entry per line.                        #
#                                                                    #
#               Case and whitespace is ignored.                      #
#               Comment lines begin with #.                          #
#                                                                    #
######################################################################

#
#   Port Types
#   ==========
        X.25
        FR
        TR
        ITI
        LAPD
        UTP
        HTDS
        SNA
        SNAPU
```

Comment out the port types that the *DPN Agent* is to ignore during network topology discovery.

For example, you can deploy the *DPN Agent* in a configuration where surveillance of only network trunks is desired (that is, surveillance of access ports is not required). In this situation, you can edit the *dpnProxy.sel* configuration file and comment out all the port types except UTP. During network topology discovery, the *DPN Agent* discards all discovered ports (except UTP ports) so that the MIB contains only the desired ports.

If the port type *SNA* is specified in the configuration file *dpnProxy.sel,* this does not result in the associated PUs being populated into the MIB, unless the port type *SNAPU* is also explicitly specified. If the port type *SNAPU* is specified in the configuration file *dpnProxy.sel,* both the SNA ports and all the associated PUs are populated into the MIB even if the port type *SNA* is not explicitly specified.

dpnProxy.map

The *DPN Agent* executes NCS commands and maps responses to SNMP variables to accomplish some of its operational features. The mapping of SNMP variables to the appropriate NCS commands is dynamically defined in the *dpnProxy.map* configuration file.

The figure "Sample dpnProxy.map" (page 38) shows a partial sample of the *dpnProxy.map* file.

WARNING
Do not modify the *dpnProxy.map* configuration file unless directed to do so by Nortel Networks support personnel.

Figure 6
Sample dpnProxy.map

```
#######################################################################
#                                                                     #
# File:         dpnProxy.map                                          #
#                                                                     #
# Description:  This file contains the SNMP variable to NCS command  #
#               mapping information and operational characteristics   #
#               of the variable relating to retrieval time.           #
#                                                                     #
# Format:       Each tuple consists of the following fields:          #
#                                                                     #
#               1.) SNMP variable name                                #
#               2.) Port type                                         #
#               3.) NCS command                                       #
#               4.) Operational flag                                  #
#                        INIT - retrieve value once at init time      #
#                        POLL - retrieve value via polling            #
#                        INFO - informational purposes only           #
#                                                                     #
#               Case and whitespace is ignored.                       #
#               Comment lines begin with #.                           #
#                                                                     #
#######################################################################

#
# SNMP variable ’ifPhysAddress’
#
ifPhysAddress:X.25:Q DNA:INIT
ifPhysAddress:ITI:Q DNA:INIT
ifPhysAddress:TR:Q PORT HARDWARE:INIT
#
# SNMP variable ’ifMtu’
#
ifMtu:ITI:Q DNA:INIT
ifMtu:FR:Q LINK:INIT
ifMtu:X.25:Q DNA:INIT
ifMtu:LAPD:Q LINK:INIT
...
```

dpnProxy.ver

This configuration file defines the version number of the *DPN Agent*.

The figure “Sample dpnProxy.ver” (page 39) shows a partial sample of the *dpnProxy.map* file.

WARNING
Do not modify the *dpnProxy.ver* configuration file unless directed to do so by Nortel Networks support personnel.

Figure 7
Sample dpnProxy.ver

```
######################################################################
#                                                                    #
# File:         dpnProxy.ver                                         #
#                                                                    #
# Description:  This file defines the software version               #
#               of the DPN SNMP Agent.                               #
#                                                                    #
# Format:       DPN SNMP Agent <version> (level)                     #
#                                                                    #
######################################################################

DPN SNMP Agent 1.0 (961001)
```

Note: The *DPN Agent* version and level numbers in the *dpnProxy.ver* file can differ with those depicted above, depending upon the specific *DPN Agent* distribution installed.

Chapter 5 Operations

This chapter provides instructions to assist you in running the *DPN Agent* and to control its operational characteristics. This includes:

The operation of the *DPN Agent* is very simple in nature and consists of the following step:

1. Execution of the *DPN Agent dpnProxy* server with the appropriate command-line options.

Components of the DPN Agents

The *DPN Agent* is composed of two executable processes that operate independently from one another:

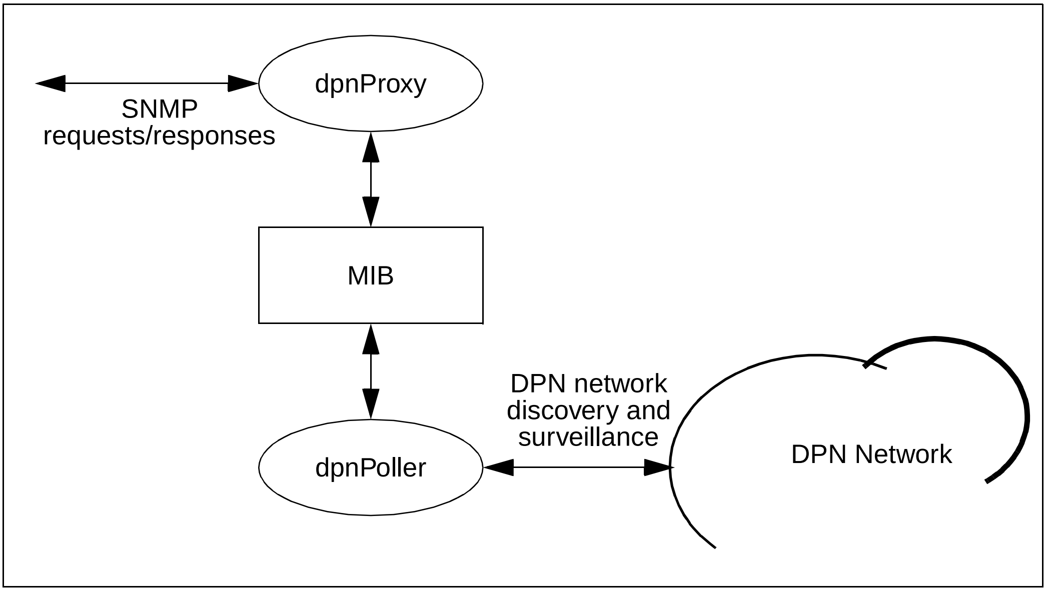
- the *dpnProxy* executable which implements an SNMP request server to accept third-party SNMP manager requests, retrieve network information from the MIB, and return the retrieved information to the requesting SNMP manager
- the *dpnPoller* executable which implements the DPN network discovery and surveillance capability of the *DPN Agent*

  The *dpnPoller* executable performs all communications with the DPN network and writes the discovered network information into the *DPN Agent* MIB.

Only the *dpnProxy* executable is launched to start the *DPN Agent*. The *dpnPoller* executable is automatically spawned by the *dpnProxy* process.

The *dpnProxy* and *dpnPoller* processes communicate with one another. Discovered DPN network information is stored within the *DPN Agent* Management Information Base (MIB) which is created dynamically at runtime by the *dpnProxy* process. The relationship between the *DPN Agent* components is shown in the figure “Logical DPN Agent components” (page 43).

Figure 8
Logical DPN Agent components



Invoking the DPN Agent

This section describes the *DPN Agent* executable and includes the following topics:

- “Distribution location” (page 46)
- “Operator command channels” (page 46)
- “Debugging and tracing” (page 47)
- “MIB storage mechanism” (page 50)
- “Host MDM” (page 51)
- “Maximum supported ports” (page 51)
- “OA hierarchy probe” (page 51)
- “SNMP port number” (page 53)
- “OS resource requirements” (page 54)
- “Port discovery mode” (page 55)

- “SNMP front-end mode” (page 60)
- “NCS traffic log” (page 61)
- “Parse failure log” (page 62)

Use the following command to invoke the *DPN Agent*:

```
[<executablePath>]dpnProxy -l <distributionPath>
[-c <commandChannels>] [-d <debugOptions>] [-f]
[-h <hostNMS>] [-i <maximumPorts>] [-o]
[-p <portNumber>] [-q] [-r] [-s] [-t] [-x]
```

where:

`[<executablePath>]`

is the optional path to the *DPN Agent dpnProxy* executable when the *DPN Agent* is invoked from a location where the *dpnProxy* executable is not in the current directory or the environment search path (for example, the system startup script)

`-l <distributionPath>`

`<distributionPath>` is the absolute path to the location of the *DPN Agent* distribution directory.

Note: This parameter is mandatory and must be the first argument specified.

`[-c <commandChannels>]`

specifies the number of operator command channels to establish to the Preside Multiservice Data Manager (MDM) environment for command processing. `<commandChannels>` is a positive, non-zero number. The default value is four.

`[-d <debugOptions>]`

directs the *DPN Agent* to generate runtime debugging and tracing information and log it to the *dpnProxy.trace* and *dpnPoller.trace* files. `<debugOptions>` is a string of trace/debug options.
The default values are FATAL,SNO,PARM,ERRORS and MAJOR.

`[-f]`

switches the *DPN Agent* into the memory-mapped file MIB implementation

`[-h <hostNMS>]`

specifies the host MDM workstation.
`<hostNMS>` is either a valid hostname or IP address.
The default value is *localhost.*

`[-i <maximumPorts>]`

is the maximum number of supported ports.
`<maximumPorts>` is a positive, non-zero number.
The default value is 5000.

`[-o]`

directs the *DPN Agent* to perform OA hierarchy discovery

Note: This option must not be executed using the MDM Server Administration tool. Use of this option must be executed from the UNIX command line. See “Server Administration” (page 67) for more information about the MDM Server Administration tool.

`[-p <portNumber>]`

is the SNMP port number for incoming requests.
`<portNumber>` is a valid port number.
The default value is 161.

`[-q]`

to query the *DPN Agent* for sizing information

Note: This option must not be executed using the MDM Server Administration tool. Use of this option must be executed from the UNIX command line. See “Server Administration” (page 67) for more information about the MDM Server Administration tool.

`[-r]`

directs the *DPN Agent* to enter port restriction mode and limit the monitored network ports to those specified in *dpnProxy.ports*

`[-s]`

directs the *DPN Agent* to enter SNMP front-end only mode in which no *dpnPoller* process is initiated

`[-t]`

directs the *DPN Agent* to log NCS traffic statistics (that is, generated traffic into and out of the DPN network and number of translated DPN alarms/logs) to the *dpnProxy.traffic* file

`[-x]`

directs the *DPN Agent* to disable logging of operator command response failures to the *dpnProxy.fail* file

Each of the above command-line arguments is examined individually in the following sections and a description of how the operation of the *DPN Agent* is affected by the option is provided.

Distribution location

The *DPN Agent* processes (that is, the *dpnProxy* and *dpnPoller* processes) use the distribution directory as the location of executables and associated configuration files. The distribution directory is also the location to which *DPN Agent* generated files (for example, *dpnProxy.ports*) are written.

The path specified by the `-l <distributionPath>` command-line option must be a UNIX absolute path and specified as the first argument to the *dpnProxy* process. The specified path should correspond to the higher-level *DPN Agent* installation directory and not one of its subdirectories (that is, *bin*, *config*, *log* or *doc*).

Operator command channels

The *DPN Agent* issues operator commands into the DPN network in order to discover the network and maintain the collected surveillance information. These operator commands are issued to a host Preside Multiservice Data

Manager (MDM) environment for processing by the servicing OA using a communications command channel. Each command channel is capable of supporting a single asynchronous operator command request/response transaction at any given time.

The throughput of operator command processing (and therefore the processing load on the servicing OA) is directly related to the number of configured operator command channels. This command line option in conjunction with “NCS traffic log” (page 61) should be used to tune the *DPN Agent* to an acceptable level of operation in relation to generated loads on servicing OAs.

The number of operator command channels is specified using the `-c <commmandChannels>` option.

The default number of operator command channels configured by the *DPN Agent* is 4. The maximum number of possible command channels is 6.

Debugging and tracing

The *DPN Agent* processes (*dpnProxy* and *dpnPoller*) generate debugging and tracing logs to log the operation of the *DPN Agent*. They also provide a means for problem identification and resolution.

The runtime tracing and debugging options are intended for the generation of execution audit logs during problem resolution investigations.

> ***Note:*** You should only executed the *DPN Agent* in this mode when you are directed to do so by Nortel Networks support personnel.

The supported `<debugOptions>` are as follows:

`FATAL`

errors (for example, memory allocation failures and socket creation failures) that are fatal to the operation of a *DPN Agent* and therefore are the cause of the *DPN Agent* shutdown

`SNO`

should not occur errors that represent unexpected or unhandled conditions encountered with by the *DPN Agent*

`PARM`

invalid or missing parameters as supplied by a user or passed into a *DPN Agent* function call

`ERRORS`

all other *non-categorized* errors as defined above

`MAJOR`

DPN Agent major events (for example, startup, connection to other servers)

`MINOR`

DPN Agent minor events

`INFO`

DPN Agent informational messages

`TRACE`

DPN Agent function call tracing

`ALL`

all debug and trace levels

To specify a debugging and tracing option, specify a comma-delimited list of one or more of the above options.

For example, to log major and minor events the command-line argument type the following command at the UNIX command line:

```
dpnProxy -d MAJOR,MINOR, <other arguments>
```

Note: Individual levels must be separated from one another by a single comma with no intervening spaces.

Both *DPN Agent* processes (*dpnProxy* and *dpnPoller*) use the same levels specified and log their generated information to the files *dpnProxy.trace* and *dpnPoller.trace* respectively. Two trace files are generated since the processes are independently executing entities.

The figure “Sample dpnProxy.trace output” (page 49) shows a sample output of the *dpnProxy.trace* file.

Figure 9
Sample dpnProxy.trace output

```
MAJOR      1995/08/12 14:36:17  dpnProxy starting up
TRACE      1995/08/12 14:36:17  Entered setupMibCacheContents()
TRACE      1995/08/12 14:36:17  Entered getSharedMemoryCache()
TRACE      1995/08/12 14:36:17  Entered getSemaphore()
INFO       1995/08/12 14:36:17  Accessing existing semaphore
TRACE      1995/08/12 14:36:17  Entered getCache()
INFO       1995/08/12 14:36:17  Accessing existing shared memory cache
MAJOR      1995/08/12 14:36:17  Attaching to shared memory cache
MAJOR      1995/08/12 14:36:17  Attached to shared memory cache
INFO       1995/08/12 14:36:17  Shared memory mapped at 0xf7210000
INFO       1995/08/12 14:36:17  Shared memory size   at 5120000 bytes
TRACE      1995/08/12 14:36:17  Entered unlockMemoryCache()
INFO       1995/08/12 14:36:17  Semaphore is not locked
TRACE      1995/08/12 14:36:17  Entered buildMibTable() with parameters
TRACE      1995/08/12 14:36:17    tableNumber = 0
TRACE      1995/08/12 14:36:17    snmpMibTableName = system
TRACE      1995/08/12 14:36:17    maxRows = 1
MAJOR      1995/08/12 14:36:17  Table entry ’system’ at 0xf7210000
...
```

If you do not override this level of debugging and tracing, using the command-line option, the *DPN Agent* processes log exceptions and major events (FATAL, SNO, PARM, ERRORS and MAJOR) to their respective log files by default. Exceptions and major events represent the minority of output of a *DPN Agent* process.

The log files generated by the *DPN Agent* can grow excessively if many options are in effect. It is suggested that *DPN Agent* processes *not* be run continuously with numerous levels in effect.

You should periodically archive and purge the *DPN Agent* log files.

WARNING
The *DPN Agent* must be properly shutdown prior to archiving or purging the trace files to avoid file access conflicts. Deletion of trace files while the *DPN Agent* is running improperly closes the files and disable logging of future events.

MIB storage mechanism

The *DPN Agent* processes (*dpnProxy* and *dpnPoller*) make use of shared data structures into which the discovered network topology information is written to and read from. These data structures represent the Management Information Base (MIB) of the *DPN Agent*. The physical implementation of the MIB is selectable at run-time. Two implementations are supported as follows:

- *shared-memory* in which the MIB is stored into, and accessed from, a shared memory cache that the *DPN Agent* requests from the workstation operating system (OS).

  This provides the most rapid MIB access since data is retrieved from in-memory structures. However, the size of the shared memory cache required by the *DPN Agent* may dictate kernel reconfiguration on the workstation that is running the *DPN Agent*.

- *memory-mapped file* in which the MIB is stored on disk but mapped into each *DPN Agent* process address space.

  This provides somewhat slower MIB access (because data is retrieved from secondary storage) but is more flexible since only disk space equivalent to the MIB size is required.

The default MIB implementation is by means of shared-memory. Specification of the `-f` command-line option switches the *DPN Agent* into the memory-mapped file implementation in which the MIB is stored on disk within the *dpnProxy.mib* file.

Refer to “OS resource requirements” (page 54) for information on sizing MIB requirements.

Host MDM

The *DPN Agent* requires the use of a host Preside Multiservice Data Manager (MDM) workstation for access to and transport of communications into the DPN network.

The host MDM workstation is specified using the `-h <hostNMS>` option.

If the *DPN Agent* is running on the same workstation as the MDM then the value *localhost* would be specified for `<hostNMS>`. The default host MDM workstation is *localhost*.

Maximum supported ports

The *DPN Agent* models DPN ports as interfaces within its MIB representation. A maximum number of supported ports must be specified in order for the *DPN Agent* to correctly determine its MIB sizing and request the appropriate system resources to realize the MIB.

Any additional ports beyond the maximum supported ports that are found during DPN topology discovery are *discarded* by the *DPN Agent*. Therefore, the network, or a portion of the network, that the *DPN Agent* is to support and survey should be properly sized. Correct port sizing ensures the proper use of OS resources.

The maximum number of supported ports is specified using the *-i <maximumPorts>* command-line option. The default maximum number of supported ports is 5000.

OA hierarchy probe

When the *DPN Agent* is invoked with the OA hierarchy probe flag, the OA hierarchy rooted at the top-level OA defined in *dpnProxy.oas* is probed and displayed to the screen. Using this information, you can modify the contents of the *dpnProxy.oas* to optimize communications with the DPN network.

Note 1: To make use of the OA hierarchy probing, at least one valid OA must be configured within the *dpnProxy.oas* configuration file so that the *DPN Agent* can access the DPN network and perform its probing operations.

Note 2: The *dpnProxy* and *dpnPoller* processes exit after the OA hierarchy has been probed.

Note 3: This option must not be executed using the MDM Server Administration tool. Use of this option must be executed from the UNIX command line. See “Server Administration” (page 67) for more information about the MDM Server Administration tool.

OA hierarchy probing is available for optimizing the OA configuration information of the *DPN Agent*. If only a single top-level OA is defined in the *dpnProxy.oas* file, all command requests are dispatched using this OA. A single configured OA is suboptimal if a large number of modules are to be supported. Since the *DPN Agent* can support communications with multiple concurrent OAs, configuration of all OAs that may need to dispatch requests is optimal. This enables more requests to be processed in a given time frame (by multiple OAs). Therefore, response time can be shortened by dispatching the request to the more immediate superior OA of a given module, rather than dispatching everything from the top-level OA down.

Refer to “dpnProxy.oas” (page 32) for information regarding the *dpnProxy.oas* configuration file.

The OA hierarchy probe flag dynamically discovers the OA hierarchy rooted at the top-level OA and report on the defined configuration information for each OA.

A sample probe, as shown in the figure “Sample OA hierarchy probe” (page 53), is of a very simple hierarchy which is rooted at CORENCS and has one subordinate OA present; GTSNWK0. The report indicates only CORENCS was configured within *dpnProxy.oas* and that the configured capability (IWSA) supports the reception of alarms but not logs. The complete hierarchy was discovered since no OAs reported a failure during subordinate discovery.

Figure 10
Sample OA hierarchy probe

```
bcars515# dpnProxy -i 1000 -h bcars424 -o

DPN SNMP Agent NCS OA Communications
====================================
The DPN SNMP Agent has completed discovering the
OA hierarchy rooted at CORENCS and has discovered
the following OAs with which communications may be required.

               DPN  Proxy                    Sub
OA Name        Capability     Parent OA     Disc.  Alarms  Logs
----------------------------------------------------------------
CORENCS        IWSA           CORENCS        yes    yes     no
GTSNWK0                       CORENCS        yes    no      no

Those OAs that could not be discovered further are indicated
above and are the result of the inability to communicate
with the OA.

Those modules which are controlled by OAs whose connect
information has not been defined in’dpnProxy.oas’
will be probed via CORENCS.

bcars515#
```

The OA hierarchy probe is a quick and convenient means by which the *DPN Agent* can investigate the OA hierarchy. This includes determining which OAs may need to be configured within *dpnProxy.oas* for optimal NCS communications, and if the defined top-level OA capability supports alarm and log reception from the DPN network.

The OA hierarchy probe functionality is meant solely as a utility for the *DPN Agent* user. Its results are not to be interpreted as the sole means by which OA communications can be configured.

SNMP port number

The *DPN Agent* requires the use of a User Datagram Protocol (UDP) port number on which to listen for and process management platform requests. The standard SNMP port number may not be suitable in those environments where some other application process is already using (listening on) the standard port number. The *DPN Agent* permits runtime configuration of the port number to satisfy varied user requirements.

The SNMP port number is specified using the *-p <portNumber>* command-line option. The default SNMP port number is 161.

OS resource requirements

The *DPN Agent* requires various OS resources depending upon the type MIB representation selected (refer to “MIB storage mechanism” (page 50)). In order to help you size these requirements, the OS resource requirements command-line flag directs the *DPN Agent* to calculate and report the size of its MIB.

> ***Note 1:*** In order to correctly calculate the size of its MIB, the *DPN Agent* requires you to specify the number of supported DPN ports at runtime (refer to “Maximum supported ports” (page 51)).
>
> ***Note 2:*** The *dpnProxy* and *dpnPoller* processes exit after the query has completed.
>
> ***Note 3:*** This option must not be executed using the MDM Server Administration tool. Use of this option must be executed from the UNIX command line. See “Server Administration” (page 67) for more information about the MDM Server Administration tool.

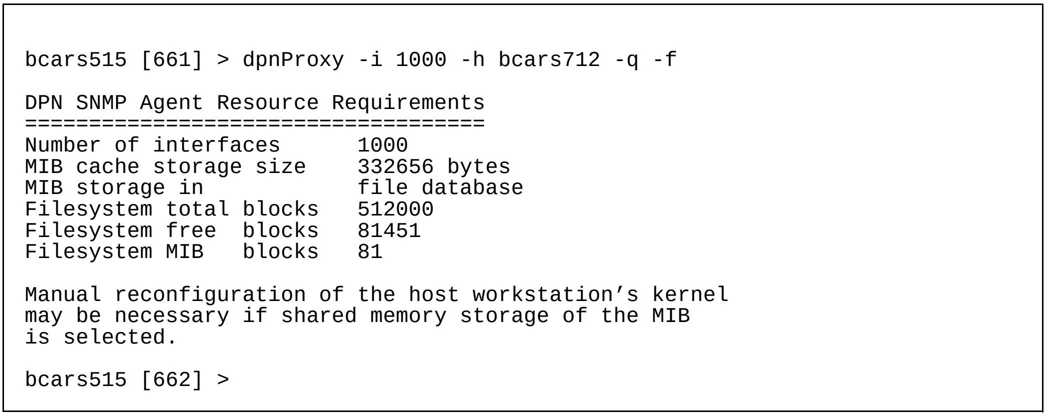
The figure “Sample resource requirements query 1” (page 54) shows a sample report where 1000 ports are required. Using the *DPN Agent* `-q` command-line option, the related sizing information is reported back.

Figure 11
Sample resource requirements query 1

```
bcars515 [660] > dpnProxy -i 1000 -h bcars712 -q

DPN SNMP Agent Resource Requirements
====================================
Number of interfaces      1000
MIB cache storage size    332656 bytes
MIB storage in            shared memory

Manual reconfiguration of the host workstation’s kernel
may be necessary if shared memory storage of the MIB
is selected.

bcars515 [661] >
```

In the figure “Sample resource requirements query 1” (page 54), the *DPN Agent* indicates that 332656 bytes of shared memory is required to represent the MIB. Shared memory storage is indicated because the default was not overridden on the command line (refer to “MIB storage mechanism” (page 50)). If it had been, the response would have been as shown in the figure “Sample resource requirements query 2” (page 55).

Figure 12
Sample resource requirements query 2



```
bcars515 [661] > dpnProxy -i 1000 -h bcars712 -q -f

DPN SNMP Agent Resource Requirements
====================================
Number of interfaces      1000
MIB cache storage size    332656 bytes
MIB storage in            file database
Filesystem total blocks   512000
Filesystem free  blocks   81451
Filesystem MIB   blocks   81

Manual reconfiguration of the host workstation’s kernel
may be necessary if shared memory storage of the MIB
is selected.

bcars515 [662] >
```

In the event the memory-mapped file database is chosen, the filesystem the *DPN Agent* is located in, total and free block counts are displayed. Also displayed are the number of blocks required to store the MIB.

Port discovery mode

The *DPN Agent* has been designed to provide you with complete control over which ports are to be monitored by the *DPN Agent* with the current surveillance set periodically written to the *dpnProxy.ports* file. This flexibility is provided by two modes in which the *DPN Agent* can be executed.

- Port *auto-discovery* mode enables the *DPN Agent* to automatically probe the network resources for member modules and associated ports. No pre-configuration is required except for defining the OA connectivity information. All modules and ports subordinate to the top-level OA are discovered and monitored.

- Port *restrictive* mode restricts the ports monitored by the *DPN Agent* to those ports predefined in the *dpnProxy.ports* configuration file. This mode of operation is useful when the *DPN Agent* is to monitor a select set of ports only.

  Note: Ports that terminate a network or trunk link are a special case when operating in *restrictive* mode. If one endpoint of a network or trunk link has been configured for monitoring (that is, included within the *dpnProxy.ports* file) and the other has not, the remote endpoint of the link is discovered and added to the *dpnProxy.ports* file, provided that the remote endpoint is visible from the top-level OA; see "dpnProxy.oas" (page 32). Therefore, a *DPN Agent* running in *restrictive* mode may discover some additional ports—the remote endpoints of ports associated with network and trunk links.

The default port discovery mode is *auto-discovery* unless overridden at runtime using with the `-r` command-line option.

If the *dpnProxy.ports* configuration file does not exist when the *DPN Agent* process is initiated, the *DPN Agent* is automatically placed into the *auto-discovery* mode of operation.

If the *dpnProxy.ports* configuration file does exist and the *DPN Agent* process is initiated in *auto-discovery* mode, then those modules and ports configured in the file are used as a starting point and the auto-discovery process updates the *dpnProxy.ports*.

Ports can also be discovered using the alarm stream that is received and interpreted from the top-level OA. Any ports that are resident on a module outside the scope of the top-level OA cannot be supported as there is no means to communicate with the host module.

In order to disable the discovery of ports through the alarm stream, the *DPN Agent* must be run in the *restrictive* mode.

While the *DPN Agent* is running in *auto-discovery* mode, it updates the *dpnProxy.ports* file every 15 minutes to reflect the current state of discovery with the set of ports and DPN modules discovered. Upon subsequent *DPN Agent* restarts, the contents of the *dpnProxy.ports* file are used to bootstrap the

DPN Agent knowledge of ports without requiring the topology discovery period before the ports are visible within the MIB. Auto-discovery still occurs to ensure all possible ports are discovered.

WARNING
The *dpnProxy.ports* configuration file reflects the current set of surveillance ports. Altering the *dpnProxy.sel* file between invocations of the *DPN Agent* may result in ports being removed from the *dpnProxy.ports* file whenever the current set of supported ports is a subset of a previous invocation.

Limiting monitored ports

If you want to limit the scope of the *DPN Agent* to a select subset of DPN ports (for example, to eliminate unsupported ports or modules and ports not of interest), some simple editing of the *dpnProxy.ports* file is required.

The editing rules for this file are as follows:

- Empty lines and comment text are ignored.
- One module or port record per line.
- A record of the following format is used to define a module of interest:

  ```
  Module: PM <moduleName>
  ```

  where:

  `<moduleName>` is the name of the module.

- A record of the following format is used to define a port of interest with optional endpoint information:

  ```
  Port: <localPortName> [Row: <rowNumber>]
  ```

  where:

  `<localPortName>` defines a port in the format

```
PM <moduleName> PE <peNumber> PI <piNumber>
PO <poNumber>
```

`<rowNumber>` refers to the row in the MIB at which the port instance should be populated.

- Port records need not be defined subordinate to their module record.
- The definition of a module record without any corresponding port records implies all ports are subordinate to the module (as determined using limited auto-discovery for the module) in either *auto-discovery* or *restrictive* modes of the *DPN Agent*.

An example of an *dpnProxy.ports* configuration file is shown in the figure “Sample dpnProxy.ports File” (page 59).

Figure 13
Sample dpnProxy.ports File

```
######################################################################
#
# File:         dpnProxy.ports
#
# Description:  This file is regenerated periodically by the DPN
#               SNMP Agent and documents the set of managed modules
#               and contained ports that are currently monitored
#               by the Agent.
#
#               Upon initiation of the dpnProxy this file is used
#               to seed the MIB and expedite the discovery of
#               desired ports.
#
#               Usage of this file in conjunction with the dpnProxy’s
#               restrictive mode flag (i.e., the -r flag) can be used
#               to configure the Agent for monitor a restricted set
#               of ports which the user may edit this file to define.
#
# Format:       This file is composed of a series of records, one
#               record per line, with the record being one of the
#               following format.
#
#               Module Record
#               =============
#               Module: PM <moduleName>
#               Example: “Module: PM R75”
#
#               Port Record
#               ===========
#               Port: <localPortName> [Row: <rowNumber>]
#               Example: “Port: PM R75 PE 1 PI 1 PO 1”
#               Example: “Port: PM R75 PE 1 PI 1 PO 1 Row:12”
#
#               Empty lines and comment lines are ignored.
#
#               Ports need not be defined subordinate
#               to their module record.
#
#               A row number may be assigned to a port record
#               to force location of the port to the specified
#               table row number.
#
#               Port records with no defined row number are
#               automatically assigned a row number of file
#               loading.
#
#               Duplicate row number assignment is not
#               permitted.
#
#               A module record with no subordinate port records
#               implies all contained ports in both auto-discovery
#               and restrictive modes of operation.
#
######################################################################

Module:PM A0320
Port: PM A0320 PE 1 PI 1 PO 1
Port: PM A0320 PE 1 PI 1 PO 2 Row: 10

Module:PM A0330
Port: PM A0330 PE 5 PI 5 PO 1 Row: 1
Port: PM A0330 PE 5 PI 5 PO 2 Row: 2
Port: PM A0330 PE 5 PI 5 PO 3 Row: 5
Port: PM A0330 PE 5 PI 5 PO 4
```

The comment text in the figure “Sample dpnProxy.ports File” (page 59) is automatically generated by the *DPN Agent* as part of its update cycle and need not be present in the initial *dpnProxy.ports* configuration file. The comment text is added when the file is next updated.

An alternative means of deriving a restricted set of ports is to initiate the *DPN Agent* process in *auto-discovery* mode, permit the *DPN Agent* to execute for at least 15 minutes to obtain an auto-discovered list of modules and ports, edit the *dpnProxy.ports* file to delete any ports and modules not of interest, and finally, rerun the *DPN Agent* in *restrictive* mode to only monitor the newly selected subset of ports in *dpnProxy.ports*. This file can also be edited manually to add known ports which have not been discovered, but which may be of interest for monitoring purposes.

> ***Note:*** Row numbers assigned to DPN ports are preserved on subsequent *DPN Agent* invocations. This permits SNMP manager applications which assign functionality based upon MIB indices to maintain the proper contexts after *DPN Agent* shutdown and restart.

SNMP front-end mode

The default behavior of the *DPN Agent* is to spawn the secondary *dpnPoller* process to initiate DPN network discovery and surveillance activities that eventually lead to the datafilling of MIB variables with DPN port information.

Specification of the `-s` command-line option switches the *DPN Agent* into the SNMP *front-end only* mode of operation in which no *dpnPoller* process is spawned.

This mode of operation is of value only when testing the communications between the SNMP manager application and the *DPN Agent*. The *DPN Agent* can be executed in this mode and SNMP communications can be verified by having the SNMP manager retrieve RFC-1213 *snmp* or *system* group variables. When running in this mode, DPN port information is not populated into the *DPN Agent* MIB.

NCS traffic log

The operation of the *DPN Agent* injects operator commands into the NCS OAs for processing and receives DPN network alarms for translation into SNMP traps. The issuing of operator commands, and reception of response data, translates into OA processing overhead and increased traffic on network links for request and response communications.

Specification of the `-t` command-line option causes the *DPN Agent* to log NCS traffic information on one minute intervals to the *dpnProxy.traffic* file. Logged information, in ASCII format, contains metrics which include the number of operator commands issued into each OA, and the number of responses received within the last interval. Traffic byte counts associated with operator command request and response processing is also provided, along with an interval count of the number of translated DPN network set alarms, clear alarms and logs.

An example *dpnProxy.traffic* log file is shown in the figure “Sample dpnProxy.traffic file” (page 61).

Figure 14
Sample dpnProxy.traffic file

```
#########################################################
#
# File Name      : dpnProxy.traffic
#
# Description    : This file is generated and updated by
#                  the DPN SNMP Proxy Agent to log NCS
#                  traffic information.
#
#                  This file is updated every minute to
#                  reflect traffic processed within the
#                  last interval.
#
#########################################################

                                                    Sent      Received        Events
Period  Timestamp                 NCS OA         Msg  Bytes  Msg  Bytes   Set   Clr   Log
-----------------------------------------------------------------------------------------
     1  Tue Oct 10 16:30:48 1995  CORENCS         91   5842   89  53157    12    14    69
     2  Tue Oct 10 16:31:48 1995  CORENCS        105   5963  104  51119    18     9    69
     3  Tue Oct 10 16:32:48 1995  CORENCS         85   4876   86  88523    14    10    76
     4  Tue Oct 10 16:33:49 1995  CORENCS         85   4832   82  63352    16    18    67
     5  Tue Oct 10 16:34:49 1995  CORENCS         41   2280   41  17018    18    14    73
     6  Tue Oct 10 16:35:49 1995  CORENCS         44   2465   44  20677    19    11    64
     7  Tue Oct 10 16:36:49 1995  CORENCS         46   2643   49  35958    16    13    68
     8  Tue Oct 10 16:37:49 1995  CORENCS         49   2830   47  58076    14     9    67
     9  Tue Oct 10 16:38:49 1995  CORENCS         57   3321   57  65201    16    12    76
    10  Tue Oct 10 16:39:49 1995  CORENCS         76   4391   76  58343    15     9    67
...
```

The data in the generated *dpnProxy.traffic* log file can be easily extracted and graphed to provide a runtime profile of the *DPN Agent* operation.

Parse failure log

The operation of the *DPN Agent* injects operator commands into the NCS OAs for processing and parses the received responses for population into the *DPN Agent* MIB. Under exception conditions (for example, an unsupported DPN generic) parsing can fail and result in the non-population of various MIB variables.

All parsing failures, by default, are logged to the *dpnProxy.fail* log file that can be used by Nortel Networks support personnel for resolution of parsing errors. Specification of the `-x` command-line option inhibits the default logging of parse errors to the *dpnProxy.fail* log file.

An example *dpnProxy.fail* log file is shown in the figure “Sample dpnProxy.fail file” (page 63).

Figure 15
Sample dpnProxy.fail file

```
#########################################################
#
# File Name      : dpnProxy.fail
#
# Description    : This file is generated and updated by
#                  the DPN SNMP Proxy Agent to log NCS
#                  operator response parsing failures.
#
#                  This file should be relayed to the
#                  appropriate Nortel Networks support
#                  personnel for resolution of any parsing
#                  errors reported by the Proxy.
#
#########################################################

********** Parse failure 00001 1995/11/10 16:58:27 ********

Failed element was UTP line speed
Failed target  was ’reported link speed =’
Failed port    was PM A8402 PE 6 PI 6 PO 2
Failed OA      was CORENCS
Failed command was ’ A8402 6 2 D’

***
utp service status

utp link state = disabled
link mode = 4 queue trunk link

local identification:
mnemonic N062; pi  6; port  2;
link type dedicated;  mid    94

statspool flag =      ON            link carrier = down
port switches =       dte           port operating mode = mux_dma
terminations =       no

modem leads : c    - i
     expect :      - on
      input :      - off
     output : off  -

1995-11-10 18:00:35 command executed
***
...
```

Correct operation of the *DPN Agent* should produce no parse errors and logging should not be disabled unless so directed by Nortel Networks support personnel.

The presence of an excessive amount of parse errors within the *dpnProxy.fail* log file is cause to contact Nortel Networks support personnel for problem investigation and resolution.

Terminating the DPN Agent

The *DPN Agent* is a daemon process and has no user interface for the normal shutdown or termination of the *DPN Agent* processes by an operator.

If the DPN Agent is being managed by the MDM Server Administration tool, use the Server Administration tool stop command to halt execution. See “Server Administration” (page 67) for more information.

If the DPN Agent is not being managed by the MDM Server Administration tool, send the SIGTERM(15) signal to the *DPN Agent dpnProxy* process using the UNIX `kill` command. For example, type the following commands:

```
ps -ef | grep dpnProxy
```

```
kill -15 <pid>
```

where `<pid>` is the process identifier indicated by the result of the `ps` command.

To deinstall the *DPN Agent*, see “De installing the DPN Agent” (page 25).

Duplicate alarm handling

As part of *DPN Agent* operations, a connection is made to the top-level OA for the receipt of DPN network alarms. If this top-level OA is configured to maintain an alarm cache then the current set of outstanding DPN alarms (SET events) is relayed to the *DPN Agent* at connection time. The presence of the top-level OA alarm cache impacts the *DPN Agent* because connections are routinely lost and re-established. This results in possible duplicate alarm events being received, translated and relayed to configured SNMP managers as traps.

For those SNMP manager platforms that are incapable of recognizing or discarding duplicate events, the *DPN Agent* implements some logic to reduce the possibility of emitting a duplicate event to a manager platform.

As DPN alarm events are received and translated, the *DPN Agent* alarm timestamp file *dpnProxy.ts* is updated with the event timestamp. Chronologically younger events than the current alarm timestamp are discarded and not relayed to the SNMP manager because the event should have already been reported. This avoids duplicate transmission.

Alarm clearing events (CLRs) and logs (LOGs) cannot be duplicated since these events are not cached within the top-level OA.

In order to flush the current active alarm set from the top-level OA, the *DPN Agent* must be shutdown, the *dpnProxy.ts* file deleted and the *DPN Agent* restarted. All currently active alarms within the top-level OA alarm cache are received and transmitted to configured SNMP managers because no *dpnProxy.ts* file is present on restart.

MIB synchronization

During the operation of the *DPN Agent*, a snapshot of the surveillance network is composed within the topology and ports discovery phases of operation (refer to "Operational model" (page 131)). After this one-time only discovery period, the *DPN Agent* relies on the surveillance network alarm stream to detect the presence of new network modules and contained ports. Upon new network component detection, the *DPN Agent* performs discovery limited to the new component for its inclusion into the MIB.

Using this methodology, the *DPN Agent* never deletes network components from its MIB because it cannot distinguish between a temporarily disabled or unreachable component and one that no longer exists. As a result, non-existent components remain in the MIB until you remove them.

In order to resynchronize the MIB, you must perform one of the following tasks depending upon the *DPN Agent*'s discovery mode of operation (refer to "Port discovery mode" (page 55)) and the type of synchronization wanted (full or selected synchronization).

DPN Agent in restricted or auto-discovery mode with selected synchronization

1 Shutdown the *DPN Agent*.

2 Edit the *dpnProxy.ports* configuration file to remove the obsolete network components and add new components.

3 Restart the *DPN Agent*.

DPN Agent in auto-discovery mode with full synchronization

1 Shutdown the *DPN Agent*.

2 Remove the *dpnProxy.ports* configuration file.

3 Restart the *DPN Agent*.

The frequency and degree in which to perform MIB synchronization is dependent upon the stability of the surveillance network (that is, the frequency in which components are added to, removed from and modified within the network). A policy should be developed after operational use of the *DPN Agent* indicates the presence of obsolete components and or absence of new components.

Multiple Agents per workstation

In those installations where it is desirable to have multiple *DPN Agents* present (for example, partitioning of the surveillance network), the *DPN Agent* architecture supports deployment of these agents to the same physical workstation, providing the following criteria is observed:

- each instance of the *DPN Agent* is installed in its own unique distribution directory
- the distribution location option (refer to “Distribution location” (page 46)) be correctly used to *home* each instance of the *DPN Agent* to its unique distribution directory

> **WARNING**
> If multiple *DPN Agent*s are *homed* to the same distribution directory then overwriting of generated trace and operations files occurs. Also, individual *DPN Agent* instances execute with identical configurations (duplicated surveillance of network resources).

- the configuration files for each *DPN Agent* instance (*dpnProxy.cnf* d*pnProxy.oas*, *dpnProxy.ports* and *dpnProxy.sel*) be tailored for partitioning of the surveillance network
- each instance of the *DPN Agent* be assigned a unique SNMP port number (refer to “SNMP port number” (page 53))

In addition to the above criteria, consideration must be given to the handling of the generated SNMP trap stream from the surveillance network. Since each instance of the *DPN Agent* connects to its configured top-level OA for alarm processing, duplicate generation of SNMP traps occurs if instances of the *DPN Agent* are configured for the same top-level OA.

To eliminate duplicate SNMP trap delivery, each *DPN Agent* must be configured for connection to a lower level OA in which there is no overlap of the surveillance alarm stream with any other *DPN Agent* instance. It is not possible to partition the surveillance of devices within a single OA across multiple *DPN Agent* instances without generating duplicate SNMP traps.

Server Administration

The *DPN Agent* can be placed under the control of the MDM Server Administration to automate the startup and monitoring of the *DPN Agent* process. Use the Server Administration `add` command for this purpose.

For information about the MDM Server Administration tool, see 241-6001-303 *Preside MDM Administrator Guide*.

To support the use of the Server Administration tool, the *dpnProxy* executable returns a compliant set of error codes. These are interpreted by the Server Administration tool to determine the appropriate failure explanatory text and appropriate action (that is, restart of the *DPN Agent* or not).

The following is an example of the DPN Agent startup command registered using the Server Administration window:

```
/opt/MagellanNMS/dpnAgent/bin/dpnProxy -l /opt/MagellanNMS/dpnAgent
```

Exit codes

The *DPN Agent* supports the exit codes listed in the table “DPN Agent exit codes” (page 68), providing a general indication of failure.

Table 2
DPN Agent exit codes

Exit Code	Description
`0`	Successful exit. Returned if the *DPN Agent* was started in OS resource query or OA hierarchy probing modes.
`51`	Memory resource error. Returned if unable to set up the MIB cache, a MIB integrity violation was detected, or unable to properly initialize the MIB ifTable and dpnIfTable rows.
`52`	Disk resource error. Returned when the *DPN Agent* is using the memory-mapped file database MIB storage method and insufficient disk space is available to store the MIB cache.
`55`	Bad command-line arguments. Returned for invalid command-line options, erroneous values for options or lack of mandatory distribution location as the first command-line option.
`56`	Bad configuration file or environment. Returned for erroneous configuration file options and/or format, invalid top-level OA capability or password, invalid distribution location, or restricted mode of operation and absence of dpnProxy.ports file.
`57`	Process fork/exec failure. Returned if unable to execute the dpnPoller process or unable to execute the dpnProxy process as a daemon.
`59`	IPC initialization failure. Returned if the *DPN Agent* is unable to register itself for IPC services (an indication that the MDM mnsd server is non-operational).
`61`	Signal exit. Returned if the *DPN Agent* was manually shutdown via the SIGTERM(15) signal.

To determine the specific cause of failure and subsequent *DPN Agent* exit, the *dpnProxy.trace* and *dpnPoller.trace* files should be consulted.

Troubleshooting

This section provides describes problems which you may encounter with the *DPN Agent*.

UDP port 161

A common problem associated with *DPN Agent* happens when another process has seized UDP port 161, the default SNMP port for *DPN Agent*. The following error is displayed.

```
ERROR <date/time> Cannot bind dpnProxy server socket
(error 125)
```

To resolve this problem, you can launch the *DPN Agent* by executing the *dpnProxy* command from the UNIX command line and specifying an alternate SNMP port number using the `-p` option; see “Invoking the DPN Agent” (page 43).

> ***Note:*** You must reconfigure the SNMP management station to accommodate any change in the *DPN Agent* configuration.

MDM *mnsd* server

The Preside Multiservice Data Manager (MDM) Multi-nodal Name Server Daemon (*mnsd*) software server is required by the *DPN Agent* and must be running in order to use the *DPN Agent*. This server provides inter-process communications (IPC) facilities that the *DPN Agent* requires for communications with MDM. See 241-6001-303 *Preside MDM Administrator Guide* for more information about the *mnsd* server.

To verify that the *mnsd* daemon is executing, enter the following command at the UNIX command line:

```
ps -ef | grep mnsd
```

If the response indicates that the *mnsd* daemon is not executing, see 241-6001-303 *Preside MDM Administrator Guide* for instructions on starting the *mnsd* server.

Appendix A
DPN Enterprise MIB

In addition to the standard MIBs implemented by the *DPN Agent*, a DPN Enterprise MIB is also supported by the *DPN Agent*. The contents of this MIB, and instructions for its use, are presented in this Appendix for the purposes of configuring the management *manager* applications. See “Contents of the MIB” (page 71) and the “MIB Specification” (page 82).

Contents of the MIB

The following sections describe:

- “DPN Enterprise traps” (page 71)
- “DPN Enterprise trap control” (page 74)
- “DPN alarm clearing” (page 74)
- “DPN interface table” (page 78)
- “Enterprise variable behavior” (page 79)
- “DPN Enterprise traps” (page 71)

DPN Enterprise traps

Corresponding to alarms and logs within the DPN environment, the DPN Enterprise MIB defines a generic DPN trap used to represent these alarms and logs within the SNMP domain.

This generic DPN trap contains a defined set of variable bindings that hold the specific alarm and log information. See:

- “Trap Type” (page 72)

- “Notification Id” (page 72)
- “Component Id” (page 72)
- “Customer Id” (page 72)
- “Reporting OA” (page 72)
- “Fault Area” (page 73)
- “Fault Code” (page 73)
- “Fault Mnemonic” (page 73)
- “Probable Cause” (page 73)
- “Severity” (page 73)
- “Operator Data” (page 73)
- “Expert Data” (page 73)
- “Comment Data” (page 73)
- “Trap Display Line” (page 73)

Trap Type

The *dpnTrapType* variable binding indicates if the trap is associated with a DPN SET, CLR or MSG condition.

Notification Id

The *dpnTrapNotificationId* variable binding contains the notification identifier associated with the alarm/log.

Component Id

The *dpnTrapCompId* variable binding contains the component identifier associated with the alarm/log.

Customer Id

The *dpnTrapCustomerId* variable binding indicates the customer identifier associated with the device emitting the alarm/log.

Reporting OA

The *dpnTrapReportingOA* variable binding indicates the DPN NCS operations agent (OA) reporting the alarm/log.

Fault Area

The *dpnTrapFaultArea* variable binding indicates the DPN area associated with the alarm/log (service data, hardware, software, security, protocol, debug, network, engineering, operations, or unclassified).

Fault Code

The *dpnTrapFaultCode* variable binding indicates the DPN fault code associated with the alarm/log and provides an index into 241-1001-506 *DPN-100 Alarm Console Indications*.

Fault Mnemonic

The *dpnTrapFaultMnemonic* variable binding identifies the DPN fault mnemonic associated with the alarm/log.

Probable Cause

The *dpnTrapProbableCause* variable binding identifies the probable cause associated with the alarm/log.

Severity

The *dpnTrapSeverity* variable binding identifies the severity associated with the alarm/log.

Operator Data

The *dpnTrapOperatorData* variable binding provides the operator data associated with the alarm/log.

Expert Data

The *dpnTrapExpertData* variable binding provides the expert data associated with the alarm/log.

Comment Data

The *dpnTrapCommentData* variable binding provides the comment data associated with the alarm/log.

Trap Display Line

The *dpnTrapTextual* variable binding provides a textual (ASCII) single line description of the alarm/log in the traditional DPN Integrated Alarm Display (IAD) format. The fault code, fault area, fault mnemonic, severity and fault type are provided in a format such as the following:

```
    10094025  SOFTWARE      FAILED  MAJOR     SET
```

When combined with the component identifier variable binding, a complete IAD-like description of the event is provided.

This variable binding is provided to aid those third-party management platforms that are unable to translate variable binding content to textual representation.

The event/fault data carried within the individual traps is a common set of information attributes obtained from the DPN environment. Refer to 241-1001-506 *DPN-100 Alarm Console Indications* for more detailed information.

DPN Enterprise trap control

The translation and conversion of DPN alarms and logs to SNMP traps is a configurable option using the DPN Enterprise MIB. This includes:

- “Alarm Control” (page 74)
- “Log Control” (page 74)

Alarm Control

The *dpnEnableAlarms* MIB variable enables/disables the conversion of DPN alarms with SET and CLR trap types to *DPN Agent* traps. When disabled, DPN alarm conditions are not translated and therefore cannot be forwarded to SNMP manager platforms.

Log Control

The *dpnEnableLogs* MIB variable enables/disables the conversion of DPN MSG events to *DPN Agent* traps. When disabled, DPN logs are not translated and therefore cannot be forwarded to SNMP manager platforms.

DPN alarm clearing

The *DPN Agent* provides an alarm clearing interface to permit third-party management platforms alarm clearing capabilities within the DPN network. This includes:

- “Clearing Criteria” (page 75)
- “Clearing Control” (page 75)
- “Clearing Result” (page 76)

- “Proxy Clearing” (page 77)

Alarm conditions cleared by the *DPN Agent* are physically cleared from the DPN NCS OA active alarm lists (AALs) and are removed from the NCS environment.

Cleared alarm conditions automatically generate a DPN enterprise trap corresponding to the original alarming conditions but with a CLR trap type. This permits the third-party management platform to maintain a synchronized active alarm list by removing the alarm event identified by the generated CLR alarm.

> ***Note:*** The mechanism by which the third-party management platform examines incoming DPN CLR trap events and removes identified DPN SET traps from its active alarm view is management platform specific. The *DPN Agent* provides a generic alarm clearing interface which can be potentially used by any management platform.

To access the DPN alarm clearing interface of the *DPN Agent* the management platform must be capable of issuing SNMP SET commands against the DPN Enterprise MIB.

To perform an alarm clearing, set the alarm clearing criteria attributes to values which identify the alarm that is to be cleared. Once these criteria attributes are datafilled, the clearing operation is initiated by triggering the clearing control MIB variable.

Clearing Criteria

The DPN alarm clearing interface provides four MIB variables, which the management platform must set, to uniquely identify the alarm that is to be cleared from the DPN NCS environment.

The clearing criteria attributes are *dpnClearCompId*, *dpnClearFaultCode*, *dpnClearNotificationId*, and *dpnClearReportingOA*. They should be set to values as reported within the original DPN trap variable bindings (refer to “DPN Enterprise traps” (page 71)).

Clearing Control

DPN alarm clearing control is by way of the *dpnClearAlarm* MIB variable. When the interface is available for an alarm clearing operation, the variable contains the *available* value. During an alarm clearing operation the variable contains the *busy* value. Having datafilled the alarm clearing criteria MIB variables, the management platform initiates alarm clearing by setting the *dpnClearAlarm* variable to *initiate* while it is in the *available* state.

> ***Note:*** The alarm clearing interface of the *DPN Agent* is synchronous in nature which means that only one alarm clearing operation can be performed at a time and another cannot be initiated until the current clearing operation has completed. The alarm clearing criteria attributes cannot be set while alarm clearing is underway.

Once the alarm clearing has completed, the interface moves back to the *available* state and the result of the just completed clearing operation is available for inspection within the *dpnClearResult* MIB variable.

Clearing Result

The result of an alarm clearing operation is provided within the *dpnClearResult* MIB variable; the values are:

none indicates that no alarm clearing operation has been requested; therefore, no result is available for a previous operation

inProgress indicates the current clearing operation is still ongoing

successfullNcsClear indicates the alarm was found and cleared within the DPN NCS environment

successfullProxyClear indicates the alarm was not found within the DPN NCS environment but was proxy cleared by the *DPN Agent* at the user's request

failedAlarmNotFound indicates the alarm was not found within the DPN NCS environment and could not be cleared because the user has not enabled *DPN Agent* proxy clearing

invalidCompId indicates that the clearing criteria *dpnClearCompId* MIB variable contains an illegal value

invalidFaultCode indicates that the clearing criteria *dpnClearFaultCode* MIB variable contains an illegal value

invalidNotificationId indicates that the clearing criteria *dpnClearNotificationId* MIB variable contains an illegal value

invalidReportingOA indicates that the clearing criteria *dpnClearReportingOA* MIB variable contains an illegal value

> ***Note:*** NCS alarm clearing can fail because some conditions (usually hardware related) cannot be cleared without physical corrective action. In these cases, a CLR is still generated by NCS but a failure is indicated within the *dpnTrapFaultMnemonic* trap variable binding. The alarm clearing interface returns a successful indication (since the failure to clear occurs within NCS and is unknown by the *DPN Agent*); therefore, the management platform should examine the contents of the CLR trap to determine if alarm clearing has failed.

The *dpnClearResult* MIB variable reflects the result of the previous alarm clearing operation with the exception of the **none** value (the initial value of *dpnClearResult*) which indicates that no alarm clearing operations have been requested. After an alarm clearing request has been initiated and until its completion, the *dpnClearResult* variable contains the value **inProgress**.

Proxy Clearing

DPN NCS OAs maintain an active alarm list (AAL) from which alarms are cleared. The presence of an AAL and its size are configurable within NCS and vary from site to site.

During alarm clearing, the alarm may not be found within the OA AAL and therefore cannot be cleared. An alarm may not be found within an AAL if an AAL has not been configured within NCS, or the original alarm has been discarded due to operational constraints within the OA. In either case, the alarm cannot be cleared within the DPN NCS environment; therefore, no CLR trap can be generated and delivered to the management platform.

To enable alarm clearing in these situations, the *DPN Agent* is capable of generating a proxy CLR trap when it has determined that NCS cannot do so. The enabling/disabling of the *DPN Agent* proxy clearing capability is controlled using the *dpnProxyClear* MIB variable.

DPN interface table

In addition to the DPN port information found within the standard RFC-1213 *ifTable* management object, the *DPN Agent* supports DPN-specific port information within the DPN Enterprise MIB *dpnIfTable* management object.

> ***Note:*** The statistics-related variables found within the *dpnIfTable* management object are disabled within the standard SNMP *ifTable* management object.

The variables currently supported within the *dpnIfTable* management object are as follows:

`dpnIfIndex`

which indicates the index of the interface that is currently collecting real-time statistic information (this index is correlated to the value of *ifIndex* within the standard *ifTable* management object)

`dpnIfType`

which indicates the specific DPN port type associated with the interface (implemented because the standard RFC-1213 *ifTable ifType* variable cannot always uniquely represent DPN port types; for example, UTP and ITI port types are both *propPointToPointSerial(22)*)

`dpnIfRemoteIndex`

which indicates the index of the interface that is remote endpoint for DPN trunk ports

`dpnIfPollStatus`

which indicates the state of statistics collection for the interface

`dpnIfStatisticsAge`

which specifies the *age* of the current statistic variable values

`dpnIfInOctets`

which corresponds to *ifInOctets* within *ifTable* `dpnIfInUcastPkts`

which corresponds to *ifInUcastPkts* within *ifTable*

`dpnIfInNUcastPkts`

which corresponds to *ifInNUcastPkts* within *ifTable*

`dpnIfInDiscards`

which corresponds to *ifInDiscards* within *ifTable*

`dpnIfInErrors`

which corresponds to *ifInErrors* within *ifTable*

`dpnIfInUnknownProtos`

which corresponds to *ifInUnknownProtos* within *ifTable*

`dpnIfOutOctets`

which corresponds to *ifOutOctets* within *ifTable*

`dpnIfOutUcastPkts`

which corresponds to *ifOutUcastPkts* within *ifTable*

`dpnIfOutNUcastPkts`

which corresponds to *ifOutNUcastPkts* within *ifTable*

`dpnIfOutDiscards`

which corresponds to *ifOutDiscards* within *ifTable*

`dpnIfOutErrors`

which corresponds to *ifOutErrors* within *ifTable*

Enterprise variable behavior

The contents of *dpnIfTable* rows consist of the static port information (the *dpnIfIndex, dpnIfType, dpnIfRemoteIndex, dpnIfPollStatus*) and the remaining dynamic statistics-related variables. The static variables are always present within rows and can be retrieved at any time. However, the dynamic statistics variables are only present when statistics collection has been enabled for the specific port.

To enable statistics collection an a port, an SNMP SET request must be generated to the *DPN Agent* enterprise variable *dpnIfPollStatus*. For example, to enable statistics collection on port index 2, the variable *dpnIfPollStatus.2* must be set to the value *enabled(1).* By default, statistics collection for ports is disabled.

Statistics collection is disabled by default since every additional port that is enabled injects additional operator commands into the DPN network for statistics calculation. These additional operator commands are injected indefinitely, on a best-effort basis, until statistics collection is cancelled. Accordingly, enabling statistics collection on a large number of ports generates continuous loading on network OAs which may or may not be desired.

Port statistic collection is not maintained across *DPN Agent* invocations (that is, for each and every invocation all ports are defaulted to disabled collection).

Agent operation counters and traps

The *DPN Agent* provides internal instrumentation of various counters and traps to track operational conditions encountered by the *DPN Agent* during run-time. This includes:

- “Event Type” (page 81)
- “Component Id” (page 82)
- “Trap Display Line” (page 82)

The exception counters are contained within the *dpnAgent* DPN Enterprise MIB variable group.

The exception counters currently supported within the *dpnAgent* variable group are as follows:

> `dpnAgentMibFullDiscards`
>
> which contains a count of the number of ports discarded because no rows are available within the *ifTable* and *dpnIfTable* MIB tables; a non-zero value indicates the need to augment the number of supported MIB ports (see “Maximum supported ports” (page 51))
>
> `dpnAgentPortUnsupportedDiscards`
>
> which contains a count of the number of ports discarded because the DPN port type is currently not supported by the *DPN Agent*; currently, the *DPN Agent* supports LAPB/X.25, LAPD/X.25, ITI, UTP, token-ring, HTDS, SNA XPAD, and frame relay ports
>
> `dpnAgentPortParseErrors`
>
> which contains a count of the number of parsing errors detected during *DPN Agent* operator command processing (the parsing error details are logged to the *dpnProxy.fail* log file); a large number of errors indicate that the appropriate Nortel Networks support personnel should be contacted to investigate and/or resolve the errors

In addition to incrementing the above counters during exception processing, the *DPN Agent* also emits the *dpnAgentEventTrap* trap to all configured managers to report the event. The event trap contains the following variable bindings.

Event Type

The *dpnAgentEventType* variable binding indicates the type of event being reported; the values are:

dpnAgentProcessStart indicates the initiation of either the *DPN Agent dpnProxy* or *dpnPoller* process

dpnAgentProcessEnd indicates the shutdown of either the *DPN Agent dpnProxy* or *dpnPoller* process

portDiscardMibFull indicates a discarded port due to MIB capacity problems

portDiscardUnsupported indicates a discarded port due to an unsupported port type

portParseError indicates an operator command parsing error and potential loss of MIB information

Component Id

The *dpnTrapCompId* variable binding contains the complete name of the port that experienced the *portDiscardMibFull*, *portDiscardUnsupported* or *portParseError* event. In the case of either the *dpnAgentProcessStart* or *dpnAgentProcessEnd* events, this variable binding provides the name of the *DPN Agent* process that experienced the event (that is, *dpnProxy* or *dpnPoller*).

Trap Display Line

The *dpnTrapTextual* variable binding provides a textual (ASCII) single line description of the event in the traditional DPN Integrated Alarm Display (IAD) format. The format of this variable binding for the currently supported set of event types are as follows:

```
DPNAGENT  OPERATIONS    INITIATE            MSG

DPNAGENT  OPERATIONS    SHUTDOWN            MSG

DPNAGENT  SOFTWARE      MIB FULL  DEGRADE   MSG

DPNAGENT  SOFTWARE      PARSE ER  DEGRADE   NSG

DPNAGENT  SOFTWARE      UNSUPPRT  DEGRADE   MSG
```

When combined with the component identifier variable binding, a complete IAD-like description of the event is provided and permits the display and handling of *DPN Agent* operating events in the same manner as DPN alarms and logs.

MIB Specification

This section contains the complete specification of the DPN Enterprise MIB.

```
DPN-MIB


DEFINITIONS ::= BEGIN
```

```
   IMPORTS
      OBJECT-TYPE
         FROM RFC-1212
      TRAP-TYPE
         FROM RFC-1215
      DisplayString
         FROM RFC1213-MIB
      enterprises, Counter, TimeTicks
         FROM RFC1155-SMI;
   nortel
      OBJECT IDENTIFIER ::= { enterprises 562 }
   magellan
      OBJECT IDENTIFIER ::= { nortel 2 }
   dpn
      OBJECT IDENTIFIER ::= { magellan 8 }
   dpnTrapAttributes
      OBJECT IDENTIFIER ::= { dpn 1 }
   dpnClearingAttributes
      OBJECT IDENTIFIER ::= { dpn 2 }
   dpnTrapControl
      OBJECT IDENTIFIER ::= { dpn 3 }
   dpnInterfaces
      OBJECT IDENTIFIER ::= { dpn 4 }
   dpnAgent
      OBJECT IDENTIFIER ::= { dpn 5 }


--
-- Definition of the DPN Enterprise trap attribute set.
-- These attributes define the contents of the DPN Enterprise traps.

--


   dpnTrapType OBJECT-TYPE

      SYNTAX INTEGER {

         message(1),

         set(2),

         clear(3)

      }

      ACCESS read-only

      STATUS mandatory

      DESCRIPTION

         “This attribute identifies the type of fault.


         Message events should be recorded but no associated
         clear will be sent. Transient events or traps that
         just provide information fall into this category.


         Set events should be recorded; a clear will follow
         when the condition has been corrected.


         Clear events indicate that a condition has been
         corrected.“

      ::= { dpnTrapAttributes 1 }


   dpnTrapNotificationId OBJECT-TYPE

      SYNTAX INTEGER

      ACCESS read-only

      STATUS mandatory
```

```
    DESCRIPTION
        “This attribute indicates the alarm identifier
        provided by thedevice. This identifier is unique
        within a device.“
    ::= { dpnTrapAttributes 2 }

dpnTrapCompId OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “This attribute identifies the component that is
        generating the trap.“
    ::= { dpnTrapAttributes 3 }

 dpnTrapCustomerId OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “This attribute identifies the customer which owns
        the component generating the trap.“
    ::= { dpnTrapAttributes 4 }

dpnTrapReportingOA OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION

        “This attribute identifies the name of the OA from
        which the trap was received.“

    ::= { dpnTrapAttributes 5 }


dpnTrapFaultArea OBJECT-TYPE

    SYNTAX INTEGER {

        serviceData(1),

        hardware(2),

        software(3),

        security(4),

        protocol(5),

        debug(6),

        network(7),

        engineering(8),

        operations(9),

        unclassified(10)

    }

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        “This attribute identifies the general area of the
        fault. serviceData traps are generated to indicate
        incorrect or missing service data for a DPN device.


        hardware traps are generated to indicate defective
        hardware.


        software traps are generated to indicate a software
        induced error within a DPN device.
        security traps are generated to indicate a potential
        security problem.


        protocol traps are generated to report protocol errors
        caused by entities external to the DPN equipment.


        debug traps should never be seen and are generated
        to report debugging information.


        network traps are generated to indicate problems in
        the AM/RM, the NM switch or elsewhere in the network.


        engineering traps are generated to indicate the
        incorrect engineering of a resource.


        operations traps are generated to indicate events
        caused by the operator.


        unclassified traps are generated to indicate an
        unclassified event that does not correspond to any of
        the other fault areas defined DPN Enterprise traps.“
    ::= { dpnTrapAttributes 6 }


dpnTrapFaultCode OBJECT-TYPE
    SYNTAX DisplayString (SIZE(8))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “An 8 digit code used to futher specify the alarm. The
        first 4 digits are used to identify the source of the
        alarm and the last four digits are used to identify
```

```
          the alarm. Can be used as an index into the alarm
          Nortel Networks technical publication (NTP) document which
          describes the recommended corrective action.“

     ::= { dpnTrapAttributes 7 }


dpnTrapFaultMnemonic OBJECT-TYPE

     SYNTAX DisplayString (SIZE(8))

     ACCESS read-only

     STATUS mandatory

     DESCRIPTION

          “This attribute provides some auxilliary information
          about the meaning of an alarm and is not always
          present.“

     ::= { dpnTrapAttributes 8 }


dpnTrapProbableCause OBJECT-TYPE

     SYNTAX INTEGER {

          configurationError(1),

          equipmentFailure(2),

          softwareError(3),

          otherSecurityService(4),

          communicationProtocol(5),

          debugging(6),

          communicationSubsystemFailure(7),

          underlyingResourceUnavailable(8),

          operationalCondition(9),

          probCauseUnknown(10)

     }

     ACCESS read-only
```

```
    STATUS mandatory
    DESCRIPTION
        “This attribute indicates the probable cause of the
        trap.“
    ::= { dpnTrapAttributes 9 }


dpnTrapSeverity OBJECT-TYPE
    SYNTAX INTEGER {
        ncsUnknown(1),
        ncsMinor(2),
        ncsMajor(3),
        ncsCritical(4),
        ncsDegrade(5),
        ncsOverload(6),
        ncsWildcard(7)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “This attribute identifies the severity of the
        generated trap.


        ncsUnknown indicates no (or unknown) action is
        required.


        ncsMinor indicates urgent corrective action should be
        taken to correct the problem; service to one customer
        has been affected.
        ncsMajor indicates immediate corrective action is
        required; service to one or more customers has been
        affected.


        ncsDegrade indicates no action is required (normally
        both a set and a clear will be generated); operation
        of the resource has been temporarily degraded.


        ncsOverload indicates no action is required (normally
        both a set and a clear will be generated); the
        component is experiencing an overload condition; the
        customer or network data has been destroyed because
        of a lack of capacity in the component.


        ncsWildcard indicates all alarms for this component
        are cleared."
    ::= { dpnTrapAttributes 10 }


dpnTrapOperatorData OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "This attribute provides additional information
        relating to the trap that an operator may use."
    ::= { dpnTrapAttributes 11 }


dpnTrapExpertData OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
```

```
        “This attribute provides additional information that
        may be of use to an expert if the trap cannot be
        resolved by the operator.“
    ::= { dpnTrapAttributes 12 }


dpnTrapCommentData OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “This attribute provides additional comment
        information about the fault.“
    ::= { dpnTrapAttributes 13 }


dpnTrapTextual OBJECT-TYPE
    SYNTAX DisplayString (SIZE (48))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “This attribute provides a textual concatenation
        of the fault code, fault area, the fault mnemonic,
        severity and the trap type.


        For example,


        10094025__SOFTWARE______FAILED__MAJOR_____SET
```

```
            This field in conjunction with the dpnTrapCompId
            attribute permits the composition of an IAD-like
            alarm/log display line for those management platforms
            that cannot manipulate the raw trap data to derive
            this output format.


            For example, the event browser on HP Openview NNM
            cannot convert the value 2 for the dpnTrapType field
            to the string oriented display element ‘SET’. This
            attribute may not be desired in more capable platforms
            or in those instances where an IAD-like display format
            is undesireable.“
        ::= { dpnTrapAttributes 14 }




--
-- Definition of the MIB variables to control the conversion and
transmission
-- of DPN alarms/events to the defined DPN Enterprise trap.
--


    dpnEnableAlarms OBJECT-TYPE
        SYNTAX INTEGER {
            enabled(1), -- permit conversion
            disabled(2) -- inhibit conversion
        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
        “The current state DPN alarm conversion
        (i.e., SET and CLR events) to Magellan DPN
        traps.


        enabled permits conversion of SET and CLR
        events within the Magellan DPN environment
        to SNMP traps.


        disabled prohibits event conversion with no
        resulting SNMP traps being generated.


        Default is enabled.”
    ::= {dpnTrapControl 1 }


dpnEnableLogs OBJECT-TYPE
    SYNTAX INTEGER {
        enabled(1), -- permit conversion
        disabled(2) -- inhibit conversion
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        “The current state DPN log conversion
        (i.e., MSG events) to Magellan DPN
        traps.


        enabled permits conversion of MSG
        events within the Magellan DPN environment
        to SNMP traps.


        disabled prohibits event conversion with no
        resulting SNMP traps being generated.
```

```
          Default is enabled.”
       ::= {dpnTrapControl 2 }


--
-- Definition of the generic DPN Enterprise trap corresponding to
-- events/faults (i.e., logs and alarms) within the DPN environment.
--


    dpnTrap TRAP-TYPE
       ENTERPRISEdpn
       VARIABLES{dpnTrapType,
              dpnTrapNotificationId,
              dpnTrapCompId,
              dpnTrapCustomerId,
              dpnTrapReportingOA,
              dpnTrapFaultArea,
              dpnTrapFaultCode,
              dpnTrapFaultMnemonic,
              dpnTrapProbableCause,
              dpnTrapSeverity,
              dpnTrapOperatorData,
              dpnTrapExpertData,
              dpnTrapCommentData,
              dpnTrapTextual}
       DESCRIPTION
          “This trap is generated to report events (i.e.,
          alarms and logs) within Magellan DPN network
          elements.“
```

```
      ::= 1


--
-- Definition of the alarm clearing capability/interface for
-- Magellan DPN devices.
--


--
-- Alarm Clearing Attributes
--
-- In order for an open management platform to initiate alarm clearing
-- into the Magellan DPN environment, the management platform must provide
-- the DPN SNMP Proxy Agent with four (4) pieces of information.
--
-- (1) The name of the Magellan DPN entity associated
--      with the network event to be cleared.
--      This information is associated with the
--      dpnTrapCompId trap attribute.
--
-- (2) The fault code of the desired event to be cleared.
--      This information is associated with the
--      dpnTrapFaultCode trap attribute.
--
-- (3) The notification ID of the desired event to be cleared.
--      This information is associated with the
--      dpnTrapNotificationId trap attribute.
--
-- (4) The Magellan DPN OA that reported the event.
--      This information is associated with the
--      dpnTrapReportingOA trap attribute.
--
-- These interface variables are provided below.

   dpnClearCompId OBJECT-TYPE
      SYNTAX DisplayString (SIZE (0..255))
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
         “This attribute identifies the component that is
         associated with the clearing event.“
      ::= { dpnClearingAttributes 1 }

   dpnClearFaultCode OBJECT-TYPE
      SYNTAX DisplayString (SIZE(8))
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
         “This attribute identifies the fault code that is
         associated with the clearing event.“
      ::= { dpnClearingAttributes 2 }

   dpnClearNotificationId OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
```

```
          “This attribute identifies the notification ID that is
          associated with the clearing event.“
       ::= { dpnClearingAttributes 3 }


    dpnClearReportingOA OBJECT-TYPE
       SYNTAX DisplayString (SIZE (8))
       ACCESS read-write
       STATUS mandatory
       DESCRIPTION
          “This attribute identifies the reporting OA that is
          associated with the clearing event.“
       ::= { dpnClearingAttributes 4 }


--
-- Alarm Clearing Control
--
-- Once the open management platform has datafilled the above
-- clearing interface variables, triggering of the alarm clearing action
-- is initiated by the following variable.
--


    dpnClearAlarm OBJECT-TYPE
       SYNTAX INTEGER {
          available(1),
          initiate(2),
          busy(3)
       }
       ACCESS read-write
       STATUS mandatory
```

```
      DESCRIPTION
          “This attribute controls the Magellan DPN
          alarm clearing capability.


          available indicates that the alarm clearing
          interface is available for use by a manager
          platform.


          initiate commences alarm clearing for the
          fault identified within dpnClearCompId,
          dpnClearFaultCode, dpnClearNotificationId
          and dpnClearReportingOA.


          busy indicates that the alarm clearing interface
          is currently processing a clear request.
          During this time, setting of the clearing
          criteria attributes are not permitted. The
          attribute is reset to available once the
          clear request has been processed.”
    ::= {dpnTrapControl 3 }


--

-- DPN SNMP Proxy Alarm Clearing

--

-- Normal alarm clearing, after initiation by the SNMP manager, is

-- normally handled by the NCS OA which will generate the alarm

-- clear trap (i.e., a DPN CLR event) once the alarm has been cleared.

-- However, it may be the case that NCS is incapable of generating

-- the CLR event (e.g., an AAL is not configured in the OA or the

-- original SET alarm has been discarded due to caching limitations).

-- If this is the case, no CLR event will be generated for the SNMP

-- manager. The variable below controls whether or not the Proxy
-- Agent will generate a procy clear event in those situations where
-- NCS is incapable of doing so.
--


   dpnProxyClear OBJECT-TYPE
      SYNTAX INTEGER {
         enabled(1),
         disabled(2)
      }
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
         “This attribute indicates whether or not
         the DPN SNMP Proxy Agent will generate
         a proxy alarm CLR when such a CLR cannot
         be generated by the NCS OA system.


         Default is enabled.”
   ::= {dpnTrapControl 4 }


--
-- Alarm Clearing Result
--
-- After an alarm clearing operation has been completed, the
-- result is reported within the following variable.
--

   dpnClearResult OBJECT-TYPE
```

```
        SYNTAX INTEGER {
            none(1),
            inProgress(2),
            successfullNcsClear(3),
            successfullProxyClear(4),
            failedAlarmNotFound(5),
            invalidCompId(6),
            invalidFaultCode(7),
            invalidNotificationId(8),
            invalidReportingOA(9)
        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
            “This attribute indicates the result of
            the last alarm clearing action.”
    ::= {dpnTrapControl 5 }


--
-- Magellan DPN Enterprise Interfaces table
--


-- The Interfaces table contains information on the entity’s
-- interfaces. Each interface is thought of as being
-- attached to a ‘subnetwork’. Note that this term should
-- not be confused with ‘subnet’ which refers to an
-- addressing partitioning scheme used in the Internet suite
-- of protocols.
```

```
dpnIfNumber OBJECT-TYPE

    SYNTAX INTEGER

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        “This variable identifies the number of rows
        within the DPN Enterprise dpnIfTable object.”

    ::= { dpnInterfaces 1 }


dpnIfTable OBJECT-TYPE

    SYNTAX SEQUENCE OF DPNIfEntry

    ACCESS not-accessible

    STATUS mandatory

    DESCRIPTION

        “A list of interface entries. The number of
        entries is given by the value of dpnIfPollNumber.”

    ::= { dpnInterfaces 2 }


dpnIfEntry OBJECT-TYPE

    SYNTAX DPNIfEntry

    ACCESS not-accessible

    STATUS mandatory

    DESCRIPTION

        “An interface entry containing objects at the
        subnetwork layer and below for a particular
        interface.”

    INDEX { dpnIfIndex }

    ::= { dpnIfTable 1 }
```

```
DPNIfEntry ::=
   SEQUENCE {
      dpnIfIndex
         INTEGER,
      dpnIfType
         INTEGER,
      dpnIfRemoteIndex
         INTEGER,
      dpnIfPollStatus
         INTEGER,
      dpnIfStatisticsAge
         TimeTicks,
      dpnIfInOctets
         Counter,
      dpnIfInUcastPkts
         Counter,
      dpnIfInNUcastPkts
         Counter,
      dpnIfInDiscards
         Counter,
      dpnIfInErrors
         Counter,
      dpnIfInUnknownProtos
         Counter,
      dpnIfOutOctets
         Counter,
      dpnIfOutUcastPkts
```

```
          Counter,
       dpnIfOutNUcastPkts
          Counter,
       dpnIfOutDiscards
          Counter,
       dpnIfOutErrors
          Counter
    }

dpnIfIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
       "A unique value for each interface. Its value
       ranges between 1 and the value of IfNumber. The
       value for each interface must remain constant at
       least from one re-initialization of the entity's
       network management system to the next re-
       initialization."
::= { dpnIfEntry 1 }

dpnIfType OBJECT-TYPE
    SYNTAX INTEGER {
       lapb-x25(1),
       lapd-x25(2),
       iti(3),
       frame-relay(4),
       token-ring(5),
            utp(6),

            htds(7),

            sna(8)

        }

        ACCESS read-only

        STATUS mandatory

        DESCRIPTION

            “This variable identifies the type of DPN port.”

        ::= { dpnIfEntry 2 }


    dpnIfRemoteIndex OBJECT-TYPE

        SYNTAX INTEGER

        ACCESS read-only

        STATUS mandatory

        DESCRIPTION

            “This variable identifies the index of the
            remote endpoint for UTP trunk ports.”

        ::= { dpnIfEntry 3 }


    dpnIfPollStatus OBJECT-TYPE

        SYNTAX INTEGER {

            enabled(1), -- start collecting statistics

            disabled(2) -- stop collecting statistics

        }

        ACCESS read-write

        STATUS mandatory

        DESCRIPTION

            “The current state of the collection of statistics
            for this interface.”
```

```
::= { dpnIfEntry 4 }


dpnIfStatisticsAge OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “The age of the statistics information
        for this interface.”
::= { dpnIfEntry 5 }


dpnIfInOctets OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “The total number of octets received on the
        interface, including framing characters.”
::= { dpnIfEntry 6 }


dpnIfInUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “The number of subnetwork-unicast packets
        delivered to a higher-layer protocol.”
::= { dpnIfEntry 7 }
```

```
dpnIfInNUcastPkts OBJECT-TYPE

    SYNTAX Counter

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        “The number of non-unicast (i.e., subnetwork-
        broadcast or subnetwork-multicast) packets
        delivered to a higher-layer protocol.”

::= { dpnIfEntry 8 }


dpnIfInDiscards OBJECT-TYPE

    SYNTAX Counter

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        “The number of inbound packets which were chosen
        to be discarded even though no errors had been
        detected to prevent their being deliverable to a
        higher-layer protocol. One possible reason for
        discarding such a packet could be to free up
        buffer space.”

::= { dpnIfEntry 9 }


dpnIfInErrors OBJECT-TYPE

    SYNTAX Counter

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        “The number of inbound packets that contained
        errors preventing them from being deliverable to a

        higher-layer protocol.”
```

```
::= { dpnIfEntry 10 }


dpnIfInUnknownProtos OBJECT-TYPE

   SYNTAX Counter

   ACCESS read-only

   STATUS mandatory

   DESCRIPTION

      “The number of packets received via the interface
      which were discarded because of an unknown or
      unsupported protocol.”

::= { dpnIfEntry 11 }


dpnIfOutOctets OBJECT-TYPE

   SYNTAX Counter

   ACCESS read-only

   STATUS mandatory

   DESCRIPTION

      “The total number of octets transmitted out of the
      interface, including framing characters.”

::= { dpnIfEntry 12 }


dpnIfOutUcastPkts OBJECT-TYPE

   SYNTAX Counter

   ACCESS read-only

   STATUS mandatory

   DESCRIPTION

      “The total number of packets that higher-level
      protocols requested be transmitted to a
      subnetwork-unicast address, including those that
      were discarded or not sent.”
```

```
::= { dpnIfEntry 13 }


dpnIfOutNUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “The total number of packets that higher-level
        protocols requested be transmitted to a non-
        unicast (i.e., a subnetwork-broadcast or
        subnetwork-multicast) address, including those
        that were discarded or not sent.”
::= { dpnIfEntry 14 }


dpnIfOutDiscards OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        “The number of outbound packets which were chosen
        to be discarded even though no errors had been
        detected to prevent their being transmitted. One
        possible reason for discarding such a packet could
        be to free up buffer space.”
::= { dpnIfEntry 15 }


dpnIfOutErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
```

```
          “The number of outbound packets that could not be
          transmitted because of errors.”
    ::= { dpnIfEntry 16 }


--
-- DPN SNMP Agent Operations
--


    dpnAgentMibFullDiscards OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
            “The number of ports discarded because of
            no room left within the MIB.”
    ::= { dpnAgent 1 }


    dpnAgentPortUnsupportedDiscards OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
            “The number of ports discarded because of
            unsupported port types.”
    ::= { dpnAgent 2 }


    dpnAgentPortParseErrors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory

        DESCRIPTION

            “The number of DPN operator command
            response errors encountered by the
            DPN SNMP Agent.”

    ::= { dpnAgent 3 }


    dpnAgentEventType OBJECT-TYPE

        SYNTAX INTEGER {

            dpnAgentProcessStart(1),

            dpnAgentProcessEnd(2),

            portDiscardMibFull(3),

            portDiscardUnsupported(4),

            portParseError(5)

        }

        ACCESS read-only

        STATUS mandatory

        DESCRIPTION

            “This attribute identifies the type event
            being reported by the DPN SNMP Agent.

            dpnAgentProcessStart indicates that the identified
            executable process has started up.

            dpnAgentProcessEnd indicates that the identified
            executable process has shutdown.

            portDiscardMibFull indicates that the identified
            port was discarded due to the MIB being full.

            portDiscardUnsupported indicates that the
            identified port was discarded since its port
            type is currently unsupported.
```

```
          portParseError indicates the potential loss
          of MIB information since the extracted information
          from the DPN environment could not be
          interpreted.Detailed parse error events are
          logged to the dpnProxy.fail log file unless so
          disabled.”

       ::= { dpnAgent 4 }


--

-- Definition of the generic trap emitted by the

-- agent to notify concerned SNMP managers of
-- operational exception events.

--


   dpnAgentExceptionTrap TRAP-TYPE

      ENTERPRISEdpn

      VARIABLES{dpnAgentEventType,

            dpnTrapCompId,
            dpnTrapTextual}

      DESCRIPTION

         “DPN Agent operations event trap.
         This trap is emitted to notify managers of
         DPN Agent operating events.“

      ::= 2


END
```

Appendix B
Detailed Port MIB Mappings

This appendix describes the detailed mappings from the information obtained from the DPN NCS to the appropriate *DPN Agent* SNMP MIB variables. This includes:

- “Conventions” (page 113)
- “MIB-II ifTable” (page 114)
- “DPN Enterprise dpnIfTable” (page 122)

Conventions

The tables in the following sections each contain two columns:

- **MIB Variable** identifies the name of the *DPN Agent* SNMP MIB variable for which the mapping applies
- **NCS Command/Envelope** indicates the NCS command used to obtain the information required to perform the mapping, followed in brackets by the attribute used to obtain the value, followed in some cases by a colon (:) and mappings for individual enumerated values (for example, *ifOperStatus*).

  Where *N/A* is present in this column, the semantics of the variable are such that it is not appropriate to support the variable for this type of port (for example, *ifInNUcastPackets* is not supported for any type of port for the services supported).

MIB-II ifTable

This section specifies the mappings used to obtain the SNMP *ifTable* variables from the information provided by NCS.

The scalar variable *ifNumber* is updated by the *DPN Agent* to be equal to the total number of entries in the *ifTable*.

MIB *ifTable* variables not identified in the following tables are either unsupported or have been moved into the DPN Enterprise MIB. In either case, the variables are not populated into the standard *ifTable* rows.

The table “LAPB/X.25 ifTable mappings” (page 114) identifies the *ifTable* mappings for DPN LAPB/X.25 ports.

Table 3
LAPB/X.25 ifTable mappings

MIB Variable	NCS Command/Envelope	
`ifIndex`	*DPN Agent* assigned row number.	
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*	
`ifType`	ddn-x25(4)	

`ifOperStatus`	DISPLAY (state)
	non-exist
	init
	activating
	wait-link-up
	dce restart
	ready
	dte restart
	link down
	wait link free
	deassociate
	port testing
		
		down(2)
		down(2)
		down(2)
		down(2)
		down(2)
		up(1)
		down(2)
		down(2)
		down(2)
		down(2)
		testing(3)

`ifMtu`	QUERY DNA (psize)	
`ifSpeed`	DISPLAY LINK (line speed)	
`ifAdminStatus`	up(1)	
`ifPhysAddress`	QUERY DNA (npi ddd)	

(Sheet 1 of 2)

Table 3 (continued)
LAPB/X.25 ifTable mappings

MIB Variable	NCS Command/Envelope
ifLastChange	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
ifSpecific	{0, 0}
(Sheet 2 of 2)	

The table “Frame Relay ifTable mappings” (page 115) identifies the *ifTable* mappings for DPN frame relay ports.

Table 4
Frame Relay ifTable mappings

MIB Variable	NCS Command/Envelope
ifIndex	*DPN Agent* assigned row number.
ifDescr	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*
ifType	frame-relay(32)
ifOperStatus	DISPLAY (state) non existent down(2) DNA associating down(2) allocating PVCs down(2) link down down(2) level 1 enabling down(2) level 2 disabling down(2) deactivating down(2) test in progress testing(3) ready up(1)
ifMtu	QUERY LINK (max bits in frame)
ifSpeed	QUERY LINK (line speed)
ifAdminStatus	up(1)
ifPhysAddress	N/A
ifLastChange	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
ifSpecific	{0, 0}

The table “ITI ifTable mappings” (page 116) identifies the *ifTable* mappings for DPN interactive terminal interface (ITI) ports.

Table 5
ITI ifTable mappings

MIB Variable	NCS Command/Envelope
`ifIndex`	*DPN Agent* assigned row number.
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*
`ifType`	propPointToPointSerial(22)
`ifOperStatus`	DISPLAY (state) initialized down(2) port up up(1) await user down(2) connecting up(1) auto detect down(2) connect timing down(2) connected up(1) disconnecting down(2) test in progress testing(3)
`ifMtu`	QUERY DNA (psize)
`ifSpeed`	QUERY LINK (in speed)
`ifAdminStatus`	up(1)
`ifPhysAddress`	QUERY DNA (npi ddd)
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}

The table “UTP ifTable mappings” (page 117) identifies the *ifTable* mappings for DPN universal trunk protocol (UTP) ports.

Table 6
UTP ifTable mappings

MIB Variable	NCS Command/Envelope
`ifIndex`	*DPN Agent* assigned row number.
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*
`ifType`	propPointToPointSerial(22)
`ifOperStatus`	DISPLAY (state) staging down(2) disabled down(2) under test testing(3) operational up(1)
`ifMtu`	Not supported.
`ifSpeed`	DISPLAY (reported link speed)
`ifAdminStatus`	up(1)
`ifPhysAddress`	N/A
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}

The table “Token Ring ifTable mappings” (page 117) identifies the *ifTable* mappings for DPN token ring ports.

Table 7
Token Ring ifTable mappings

MIB Variable	NCS Command/Envelope
`ifIndex`	*DPN Agent* assigned row number.
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*
`ifType`	iso88025-tokenRing(9)
(Sheet 1 of 2)	

Table 7 (continued)
Token Ring ifTable mappings

MIB Variable	NCS Command/Envelope
`ifOperStatus`	DISPLAY (state) running up(1) disabled down(2) testing testing(3) enabling up(1) disabling down(2) terminating down(2)
`ifMtu`	Not supported.
`ifSpeed`	Q PORT HARDWARE (ring speed)
`ifAdminStatus`	up(1)
`ifPhysAddress`	Q PORT HARDWARE (node address)
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}
(Sheet 2 of 2)	

The table “LAPD/X.25 ifTable mappings” (page 118) identifies the *ifTable* mappings for DPN LAPD/X.25 ports.

Table 8
LAPD/X.25 ifTable mappings

MIB Variable	NCS Command/Envelope
`ifIndex`	*DPN Agent* assigned row number.
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*
`ifType`	other(1)
`ifOperStatus`	DISPLAY (state) down down(2) normal up(1) enabling up(1) disabling down(2) unknown down(2)
(Sheet 1 of 2)	

Table 8 (continued)
LAPD/X.25 ifTable mappings

MIB Variable	NCS Command/Envelope
ifMtu	QUERY LINK (max bytes in I-field) - 1
ifSpeed	QUERY LINK (line speed)
ifAdminStatus	up(1)
ifPhysAddress	N/A
ifLastChange	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
ifSpecific	{0, 0}
(Sheet 2 of 2)	

The table “HTDS ifTable mappings” (page 119) identifies the *ifTable* mappings for DPN HTDS ports.

Table 9
HTDS ifTable mappings

MIB Variable	NCS Command/Envelope	
ifIndex	*DPN Agent* assigned row number.	
ifDescr	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*	
ifType	other(1)	
ifOperStatus	DISPLAY (state)	
	non existent	down(2)
	DNA associating	down(2)
	allocating PVCs	down(2)
	link down	down(2)
	level 1 enabling	down(2)
	level 2 disabling	down(2)
	deactivating	down(2)
	port under test	testing(3)
	ready	up(1)
ifMtu	QUERY LINK (max bits in frame)	
ifSpeed	DISPLAY LINK (line speed)	
ifAdminStatus	up(1)	
(Sheet 1 of 2)		

Table 9 (continued)
HTDS ifTable mappings

MIB Variable	NCS Command/Envelope
`ifPhysAddress`	QUERY DNA (npi ddd)
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}
(Sheet 2 of 2)	

The table “SNA ifTable mappings” (page 120) identifies the *ifTable* mappings for DPN SNA XPAD ports.

Table 10
SNA ifTable mappings

MIB Variable	NCS Command/Envelope	
`ifIndex`	*DPN Agent* assigned row number.	
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>*	
`ifType`	sdlc(17)	
`ifOperStatus`	DISPLAY (state)	
	null	down(2)
	down	down(2)
	waiting	down(2)
	wait up	down(2)
	disabled	down(2)
	wait call	down(2)
	wait call connect	down(2)
	wait down	down(2)
	wait call	down(2)
	refused	down(2)
	test in progress	testing(3)
	up	up(1)
`ifMtu`	QUERY DNA (psize)	
`ifSpeed`	DISPLAY (line speed)	
`ifAdminStatus`	up(1)	
`ifPhysAddress`	QUERY DNA (npi ddd)	
(Sheet 1 of 2)		

Table 10 (continued)
SNA ifTable mappings

MIB Variable	NCS Command/Envelope
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}
(Sheet 2 of 2)	

The table “SNA PU ifTable mappings” (page 121) identifies the *ifTable* mappings for DPN SNA Physical Units (PUs).

Table 11
SNA PU ifTable mappings

MIB Variable	NCS Command/Envelope
`ifIndex`	*DPN Agent* assigned row number.
`ifDescr`	PM *<moduleName>* PE *<peNumber>* PI *<piNumber>* PO *<poNumber>* *PU <puNumber>*
`ifType`	sdlc(17)
`ifOperStatus`	DISPLAY (state) idle down(2) contact pending down(2) sending node set down(2) sending SNRM down(2) disabled down(2) NRM primary up(1) NRM secondary up(1) sending disc down(2) sending U-frame down(2)
`ifMtu`	QUERY DNA (psize)
`ifSpeed`	matches the ifSpeed set in associated SNA port
`ifAdminStatus`	up(1)
`ifPhysAddress`	QUERY DNA (npi ddd)
`ifLastChange`	*DPN Agent* updated to sysUpTime on change of ifOperStatus.
`ifSpecific`	{0, 0}

DPN Enterprise dpnIfTable

This section specifies the mappings used to obtain the DPN Enterprise *dpnIfTable* variables from the information provided by NCS.

The scalar variable *dpnIfNumber* is updated by the *DPN Agent* to be equal to the total number of entries in the *dpnfTable*. The number of entries in the *dpnfTable* is always equal to *ifNumber.*

MIB *ifTable* variables not identified in the following tables are either unsupported or have been moved into the DPN Enterprise MIB. In either case, the variables are not populated into the standard *ifTable* rows.

The derivation of values for the *dpnfTable* statistical variables is specific to the port type and are synthesized from the collected DPN NCS statistics information (which vary depending upon port type).

The table “LAPB/X.25 dpnIfTable mappings” (page 122) identifies the *dpnIfTable* mappings for DPN LAPB/X.25 ports.

Table 12
LAPB/X.25 dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	lapb-x25(1)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY SERVICE (data bytes: received)
`dpnIfInUcastPkts`	DISPLAY SERVICE (data packets: received)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (orej+ofrej)
(Sheet 1 of 2)	

Table 12 (continued)
LAPB/X.25 dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY SERVICE (data bytes: sent)
`dpnIfOutUcastPkts`	DISPLAY SERVICE (data packets: received)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (irej+ifrej)
(Sheet 2 of 2)	

The table “Frame Relay dpnIfTable mappings” (page 123) identifies the *dpnIfTable* mappings for DPN frame relay ports.

Table 13
Frame Relay dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	frame-relay(4)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (ibyte)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (ifrm)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (bsize + proto + abort + crcer + lrcer +overw)
`dpnIfInUnknownProtos`	N/A
(Sheet 1 of 2)	

Table 13 (continued)
Frame Relay dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfOutOctets`	DISPLAY STATISTICS (obyte)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (ofrm)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (under)
(Sheet 2 of 2)	

The table “ITI dpnIfTable mappings” (page 124) identifies the *dpnIfTable* mappings for DPN interactive terminal interface (ITI) ports.

Table 14
ITI dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	iti(3)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (bytes received n+mk)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (packets received)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (overwrite + parity + security)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (bytes sent n+mk)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (packets sent)
(Sheet 1 of 2)	

Table 14 (continued)
ITI dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	N/A
(Sheet 2 of 2)	

The table “UTP dpnIfTable mappings” (page 125) identifies the *dpnIfTable* mappings for DPN universal trunk protocol (UTP) ports.

Table 15
UTP dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	utp(6)
`dpnIfRemoteIndex`	ifIndex of trunk remote endpoint.
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (receive: bytes)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (receive: frames)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (link quality: lrc + crc + overruns)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (transmitted byte: total)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (packets sent)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (link quality: underruns)

The table “Token Ring dpnIfTable mappings” (page 126) identifies the *dpnIfTable* mappings for DPN token ring ports.

Table 16
Token Ring dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	token-ring(5)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (receive: bytes)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (receive: frames)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (error counts: line + burst + lost frame + token + dma bus + ari/fci + receive congestion + frame copied + dma parity)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (transmit: bytes)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (transmit: frames)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	N/A

The table “LAPD/X.25 dpnIfTable mappings” (page 127) identifies the *dpnIfTable* mappings for DPN LAPD/X.25 ports.

Table 17
LAPD/X.25 dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	lapd-x25(2)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (ibyte)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (frecv)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (orej + ofrej)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (obyte)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (fsent)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (irej + ifrej)

The table “HDLC transparent data service (HTDS) dpnIfTable mappings” (page 128) identifies the *dpnIfTable* mappings for DPN HDLC Transparent Data Service (HTDS) ports.

Table 18
HDLC transparent data service (HTDS) dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	htds(7)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (ibyte)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (ifrm)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (bsize + proto + abort + lrcer + crcer + overw)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (obyte)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (ofrm)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (under)

The table "SNA dpnIfTable mappings" (page 129) identifies the *dpnIfTable* mappings for DPN SNA XPAD ports.

Table 19
SNA dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	sna(8)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	DISPLAY STATISTICS (sum of lengths of rus recd)
`dpnIfInUcastPkts`	DISPLAY STATISTICS (rus recd)
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (bsize + proto + abort + lrcer crcer + overw)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	DISPLAY STATISTICS (sum of lengths of rus sent)
`dpnIfOutUcastPkts`	DISPLAY STATISTICS (rus sent)
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	DISPLAY STATISTICS (under)

The table “SNA PU dpnIfTable mappings” (page 130) identifies the *dpnIfTable* mappings for DPN SNA Physical Units (PUs).

Table 20
SNA PU dpnIfTable mappings

MIB Variable	NCS Command/Envelope
`dpnIfIndex`	*DPN Agent* assigned row number equal to ifIndex.
`dpnIfType`	sna(8)
`dpnIfRemoteIndex`	N/A
`dpnIfPollStatus`	*DPN Agent* defaults to disabled(2), modifiable by manager.
`dpnIfStatisticsAge`	Age of current port statistics variables (i.e., difference between the current system time and the last update time).
`dpnIfInOctets`	N/A
`dpnIfInUcastPkts`	N/A
`dpnIfInNUcastPkts`	N/A
`dpnIfInDiscards`	N/A
`dpnIfInErrors`	DISPLAY STATISTICS (proto)
`dpnIfInUnknownProtos`	N/A
`dpnIfOutOctets`	N/A
`dpnIfOutUcastPkts`	N/A
`dpnIfOutNUcastPkts`	N/A
`dpnIfOutDiscards`	N/A
`dpnIfOutErrors`	N/A

Appendix C
Engineering measurements

This appendix provides engineering information for MIB sizing, SNMP response times, the number of operator command requests generated by the *DPN Agent* during various stages of its life cycle, and approximate byte counts for the response data. This includes:

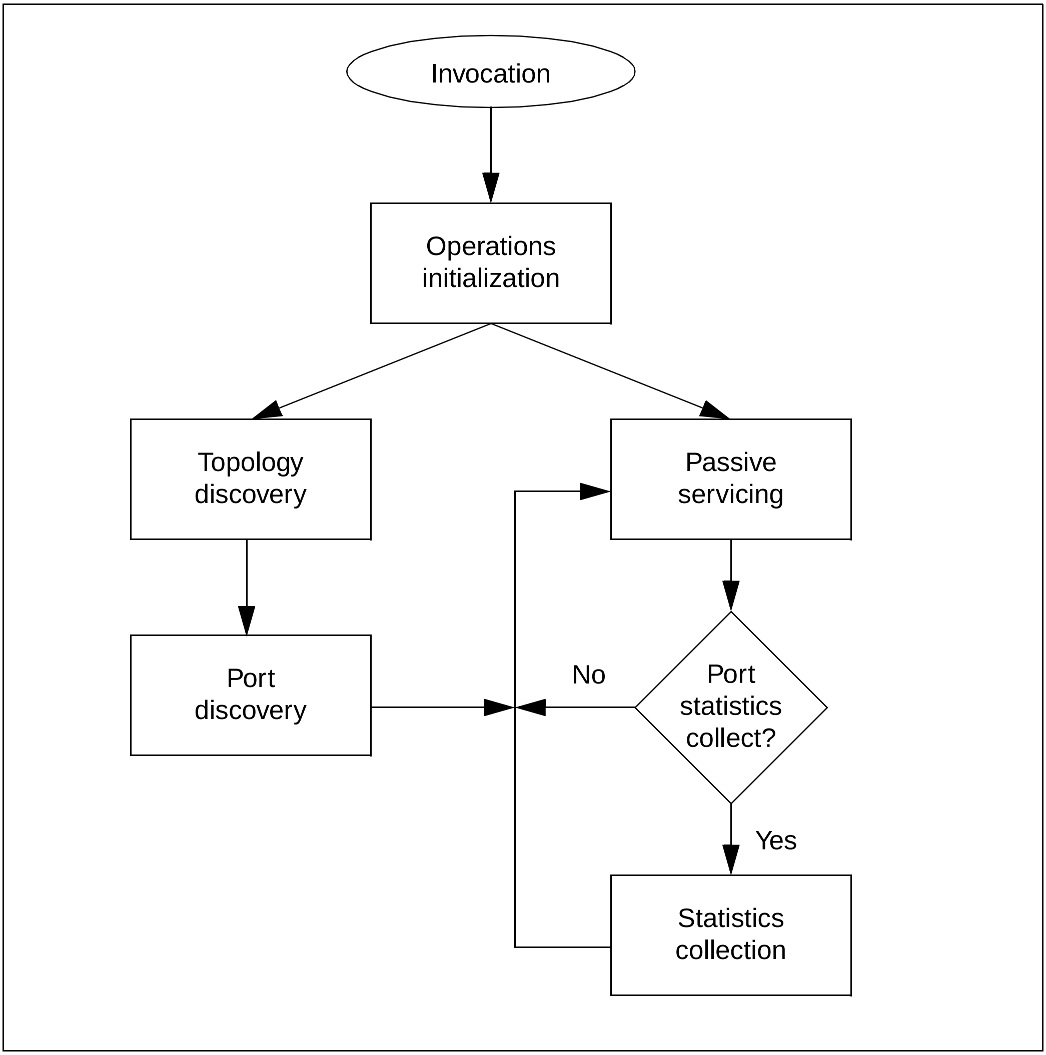
- “Operational model” (page 131)
- “MIB storage” (page 133)
- “SNMP response time” (page 134)
- “Traffic measurements” (page 134)
- “Sample configurations and measurements” (page 139)

Operational model

The life cycle of the *DPN Agent* consists of various phases, that are depicted in the figure “Agent life cycle” (page 132), are described in the following sections:

- “Operations initialization” (page 132)
- “Topology discovery” (page 133)
- “Port discovery” (page 133)
- “Passive servicing” (page 133)
- “Statistics collection” (page 133)

Figure 16
Agent life cycle



Operations initialization

Initialization consists of connecting to the various configured supporting OAs, configuring the SNMP variable to NCS command mapping tables, registering for DPN alarm reception, and various other activities to prepare the *DPN Agent* for operation. Once this phase is complete, the *DPN Agent* splits its operation into two asynchronous tasks: topology discovery and passive servicing.

Topology discovery

Topology discovery enables the *DPN Agent* to discover all of the subordinate modules to the top-level OA.

Port discovery

Port discovery enables the discovery of all network ports subordinate to discovered modules (for auto-discovery mode of operation) or those explicitly identified within *dpnProxy.ports* (for restricted mode of operation) that are to be monitored, and the one-time only collection of port configuration information and initial status information.

Passive servicing

Passive servicing represents the *DPN Agent*’s idle state where only maintenance procedures are performed (for example, module and port heart beat requests).

Statistics collection

Statistics collection occurs when and if the user indicates that statistics are to be collected (using the *dpnIfPollStatus* DPN Enterprise MIB row variable) for a set of ports. This causes the *DPN Agent* to retrieve statistical port information from the network.

Translation of DPN alarms and logs into SNMP traps and the emission of generated traps to configured SNMP managers occurs throughout the lifetime of the *DPN Agent*.

MIB storage

The *DPN Agent* provides storage for its SNMP MIB in either shared memory or a disk-based file. In either case, the storage requirements for the MIB is the same. The following formula can be used to accurately calculate the required operating system resources (that is, either shared memory RAM or filesystem disk space):

MIB size in bytes = 8980 + (324 x <maximumPorts>)

Using the above formula where <*maximumPorts*> represents the maximum number of supported ports, sample MIB storage requirements for various *DPN Agent* configurations can be calculated below.

Table 21
Sample MIB sizing requirements

Maximum ports	MIB size (bytes)
250	89 980
500	170 980
1 000	332 980
2 000	656 980
5 000	1 628 980
10 000	3 248 980

Based on the above table, the *DPN Agent* requires approximately 330 kilobytes of either shared memory RAM or filesystem disk space per 1000 supported DPN ports.

SNMP response time

The *DPN Agent* response time to SNMP manager requests is, on average, approximately 5 milliseconds regardless of configuration.

Traffic measurements

This section serves as a guide in determining the traffic requirements of the *DPN Agent* and the subsequent impact upon the monitored DPN network. This section includes

- “Operations initialization” (page 135)
- “Topology discovery” (page 135)
- “Port discovery” (page 135)
- “Passive servicing” (page 137)
- “Statistics collection” (page 138)

> ***Note:*** The message sizes indicated in the various sections of this appendix are not meant to be exact since the size of any NCS message response is somewhat variable depending upon the data it contains. However, the sizes should be accurate with plus or minus 10%. Those sizes marked *variable* are too variable in nature to even specify an approximate size.

For an accurate runtime determination of generated traffic, the *DPN Agent* should be run with NCS traffic logging (refer to “NCS traffic log” (page 61)).

Operations initialization

No traffic is generated into the network at this phase.

Topology discovery

The following requests are generated into the network depending upon device type.

Table 22
Messages per device for topology discovery

Device type	Generated request	Response size (bytes)	Total response size (bytes)
OA	DIR	variable	variable
AM	CONSOLE CONTROL Q DIR QUERY * QUERY MODULE	442 2911 446	3799
RM	CONSOLE CONTROL Q DIR QUERY * QUERY MODULE QUERY SWITCH	442 2911 446 234	4033

Port discovery

Requests generated into the network for port discovery fall into two categories:

- determining the existence of a given port
- determining the type and initial status of the port

In the case of restrictive mode, the individual modules are not interrogated for their contained ports since only the ports identified in *dpnProxy.ports* are subject to surveillance. In this case, requests into the network for determination of port existence are not required.

> ***Note:*** The only exception to this is the case where the *DPN Agent* is invoked in restrictive mode and a complete module has been designated for monitoring (refer to “Limiting monitored ports” (page 57)) by not specifying any contained ports. In this one case, the module is interrogated for all contained ports as is done for auto-discovery mode).

In the case of auto-discovery mode (and the one exception identified above) modules are interrogated for all of their contained ports by the messages defined below.

> ***Note:*** The following traffic is generated for each and every peripheral interface (PI) within the module in question.

Table 23
Messages per PI for port discovery

Device type	Generated request	Response size (bytes)	Total message size (bytes)
PI	DISPLAY PORTS	300	300

Having discovered the existence of a port, the *DPN Agent* performs the following queries on a port basis for the determination of the port type, status, and initial configuration information (for example, a one-time determination of the *ifPhysAddress* variable).

Table 24
Messages per port for configuration information

Port type	Generated request	Response size (bytes)	Total response size (bytes)
ITI	QUERY LINK QUERY DNA	409 1208	1617
X.25	QUERY DNA	1509	1509
UTP	QUERY LINK	832	832
FR	QUERY LINK	340	340
TK	QUERY PORT HARDWARE	276	276
LAPD	QUERY LINK	1500	1500

Table 25
Messages per port for port type determination

Port type	Generated request	Response size (bytes)	Total response size (bytes)
ITI	DISPLAY	190	190
X.25	DISPLAY	575	575
UTP	DISPLAY	729	729
FR	DISPLAY	231	231
TK	DISPLAY DISPLAY STATISTICS	123 913	1036
LAPD	DISPLAY	500	500

Passive servicing

In passive servicing, the *DPN Agent* performs maintenance tasks that can generate requests into the DPN network for servicing.

Every 15 minutes, the *DPN Agent* probes the status of all disabled ports. This permits the *DPN Agent* to observe ports that make the transition from operationally disabled to enabled. To effect this, the *DPN Agent* generates the same traffic identified in the figure “Traffic for 500 port network” (page 143) *for currently disabled ports only.*

Every 15 minutes, the *DPN Agent* probes the status of all monitored modules. This permits the *DPN Agent* to observe modules transition from enabled to disabled states. To effect this, the *DPN Agent* generates the following traffic for all modules currently known to it.

Table 26
Messages per module heart beat

Device type	Generated request	Response size (bytes)	Total response size (bytes)
PM	DISPLAY	475	475

Alarms and logs received from modules (PMs), processing elements (PEs), peripheral interfaces (PIs), or port (PO) components that could indicate a change in status or operational state provoke the *DPN Agent* into generating a port heart beat message for contained ports in the module identified in the alarm or log. To effect this, the *DPN Agent* generates the same traffic identified in the figure “Commands for 1000 port network” (page 144) *for all monitored ports of the indicated module*.

Statistics collection

By enabling statistics collection on ports (using the DPN Enterprise MIB variable *dpnIfPollStatus*) the collection of statistical information can be controlled on a row-by-row basis. If a row is undergoing statistics collection, the following traffic is generated into the network.

Table 27
Messages per port for port statistics collection

Port type	Generated request	Response size (bytes)	Total response size (bytes)
ITI	DISPLAY STATISTICS	189	189
X.25	DISPLAY SERVICE DISPLAY STATISTICS	876 1536	2412
UTP	DISPLAY STATISTICS	720	720
FR	DISPLAY STATISTICS	703	703
TK	DISPLAY STATISTICS	913	913
LAPD	DISPLAY STATISTICS	1375	1375

Statistics collection is reinitiated upon previous response completion while the *dpnIfPollStatus* variable indicates that statistics are to be collected.

Sample configurations and measurements

This section provides sample configurations of the *DPN Agent* and observed operating characteristics. These samples are not representative of all possible configurations and the resulting operating characteristics vary from network to network.

These sample configurations are for informational purposes only with the graphs of issued operator commands and resulting network traffic generated directly from the *DPN Agent* traffic logs (refer to “NCS traffic log” (page 61)).

All sample configurations consisted of a connection to a single top-level OA with the default number of operator command channels (that is, four). The following examples are provided:

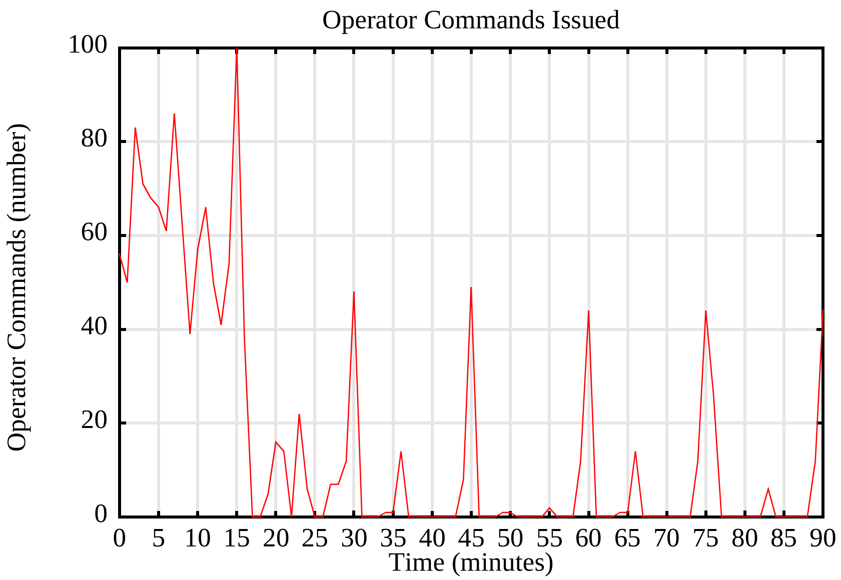
- “250 port network” (page 140)
- “500 port network” (page 142)
- “1000 port network” (page 144)
- “3000 port network” (page 146)

- “Overall observations” (page 148)

250 port network

The figure “Commands for 250 port network” (page 140) shows the number of operator commands issued to the DPN network for a 90 minute period to support surveillance of 250 ports.

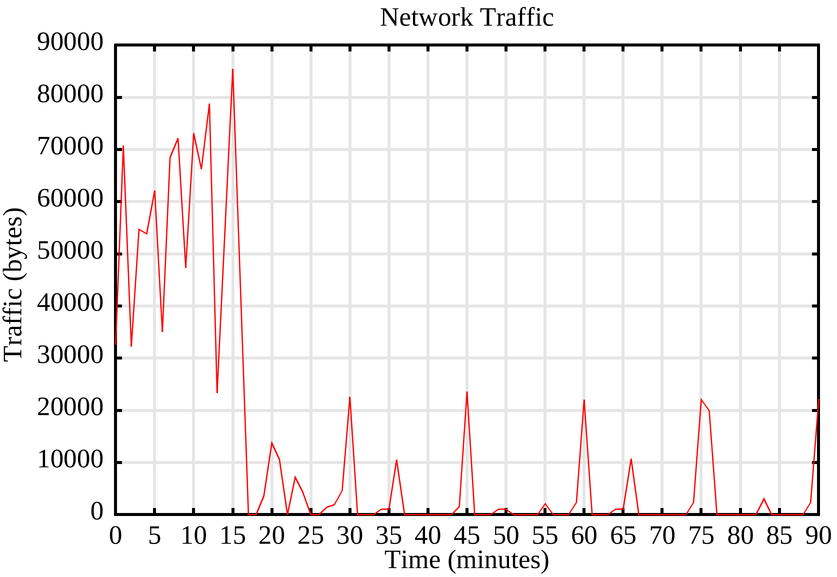
Figure 17
Commands for 250 port network



As can be seen above in the figure “Commands for 250 port network” (page 140), the *DPN Agent* concludes the topology and port discovery phases of operation after approximately 17 minutes at which point passive servicing begins with its characteristic spiking on 15 minute boundaries.

The network traffic generated to transmit the operator commands and receive the results during this period is shown in the figure “Traffic for 250 port network” (page 141).

Figure 18
Traffic for 250 port network



During topology and port discovery, the *DPN Agent*, on average, generated approximately 50 to 60 kilobytes of network traffic per minute. Upon entering the passive servicing phase of operations, traffic reduced to approximately 25 kilobytes per 15 minute interval.

The table “24 hour observations for 250 ports” (page 141) documents *DPN Agent* process characteristics for a 24 hour operating period.

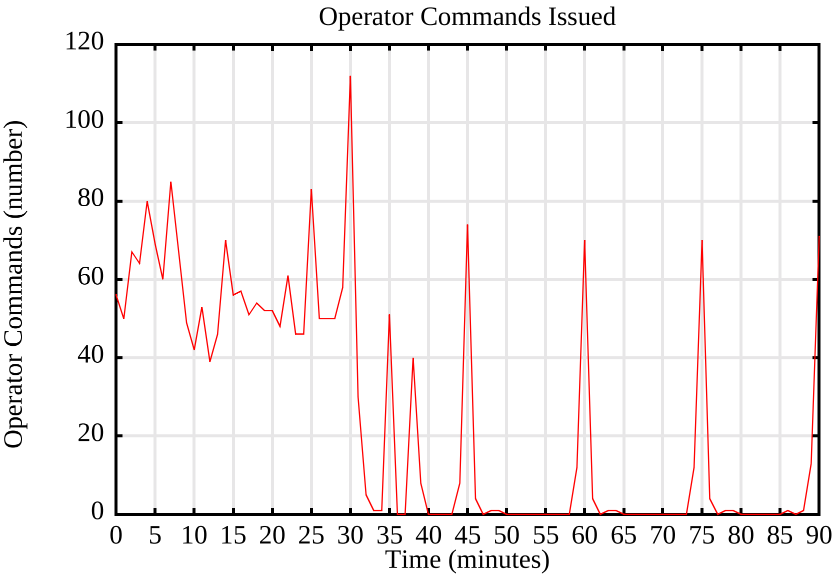
Table 28
24 hour observations for 250 ports

Criteria	Measurement
CPU Usage	dpnProxy (14 seconds) dpnPoller (7 minutes, 50 seconds)
Process Size	dpnProxy (324 kilobytes) dpnPoller (820 kilobytes)

500 port network

The figure “Commands for 500 port network” (page 142) shows the number of operator commands issued into the DPN network for a 90 minute period to support surveillance of 500 ports.

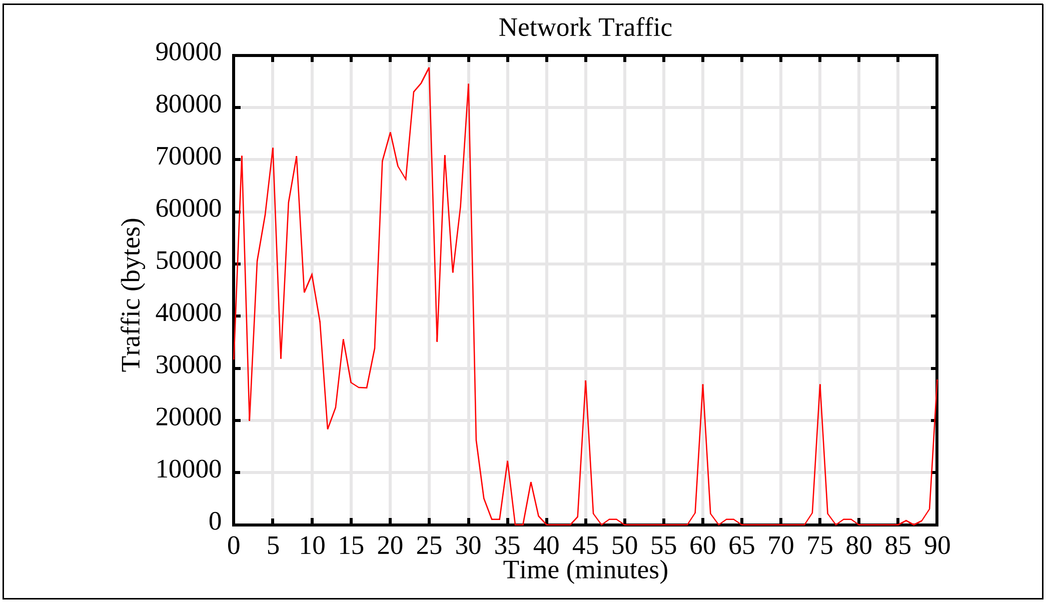
Figure 19
Commands for 500 port network



In the figure “Commands for 500 port network” (page 142), the *DPN Agent* concludes the topology and port discovery phases of operation after approximately 33 minutes at which point passive servicing begins with its characteristic spiking on 15 minute boundaries.

The network traffic generated to transmit the operator commands and receive the results during this period is provided in the figure “Traffic for 500 port network” (page 143).

Figure 20
Traffic for 500 port network



During topology and port discovery, the *DPN Agent*, on average, generated approximately 50 to 60 kilobytes of network traffic per minute. Upon entering the passive servicing phase of operations, traffic reduced to approximately 28 kilobytes per 15 minute interval.

The table “24 hour observations for 500 ports” (page 143) documents *DPN Agent* process characteristics for a 24 hour operating period.

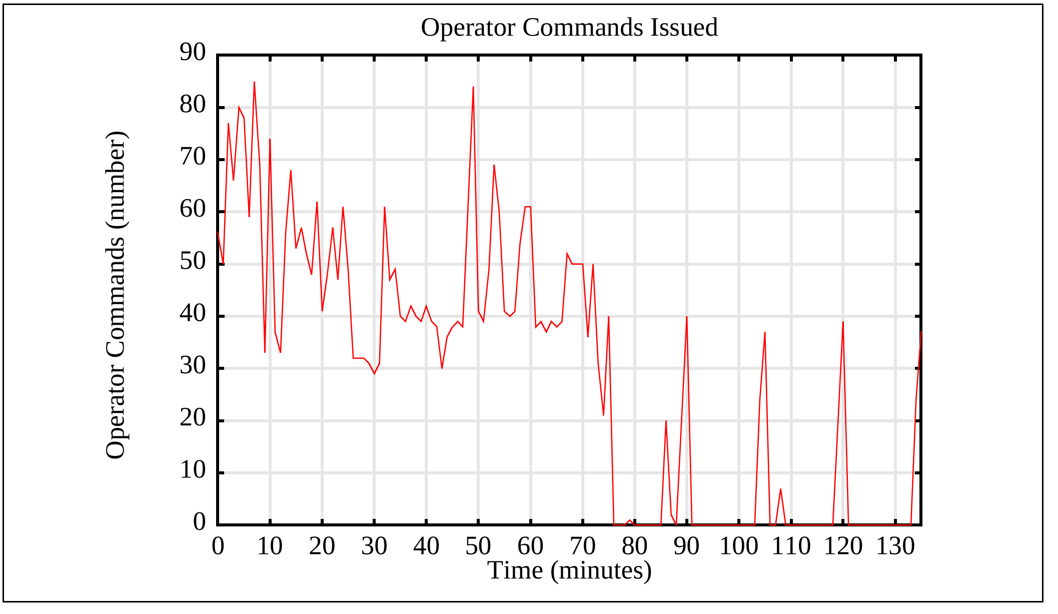
Table 29
24 hour observations for 500 ports

Criteria	Measurement
CPU Usage	dpnProxy (14 seconds) dpnPoller (5 minutes, 24 seconds)
Process Size	dpnProxy (324 kilobytes) dpnPoller (880 kilobytes)

1000 port network

The figure “Commands for 1000 port network” (page 144) shows the number of operator commands issued to the DPN network for a 136 minute period (extended from the 250 and 500 port samples due to lengthened topology and port discovery phases) to support surveillance of 1000 ports.

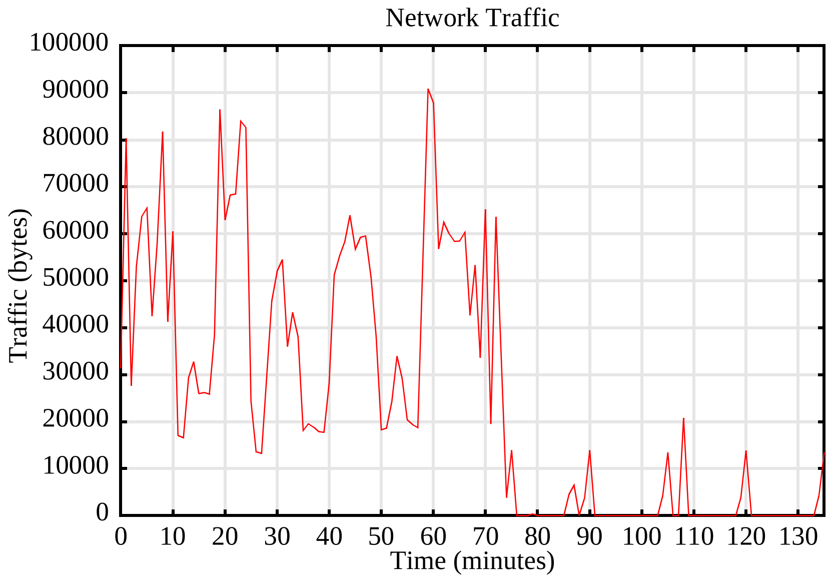
Figure 21
Commands for 1000 port network



In the figure “Commands for 1000 port network” (page 144), the *DPN Agent* concludes the topology and port discovery phases of operation after approximately 75 minutes at which point passive servicing begins with its characteristic spiking on 15 minute boundaries.

The network traffic generated to transmit the operator commands and receive the results during this period is shown in the figure “Traffic for 1000 port network” (page 145).

Figure 22
Traffic for 1000 port network



During topology and port discovery, the *DPN Agent*, on average, generated approximately 50 to 60 kilobytes of network traffic per minute. Upon entering the passive servicing phase of operations, traffic reduced to approximately 28 kilobytes per 15 minute interval.

The table “24 hour observations for 1000 ports” (page 145) documents *DPN Agent* process characteristics for a 24 hour operating period.

Table 30
24 hour observations for 1000 ports

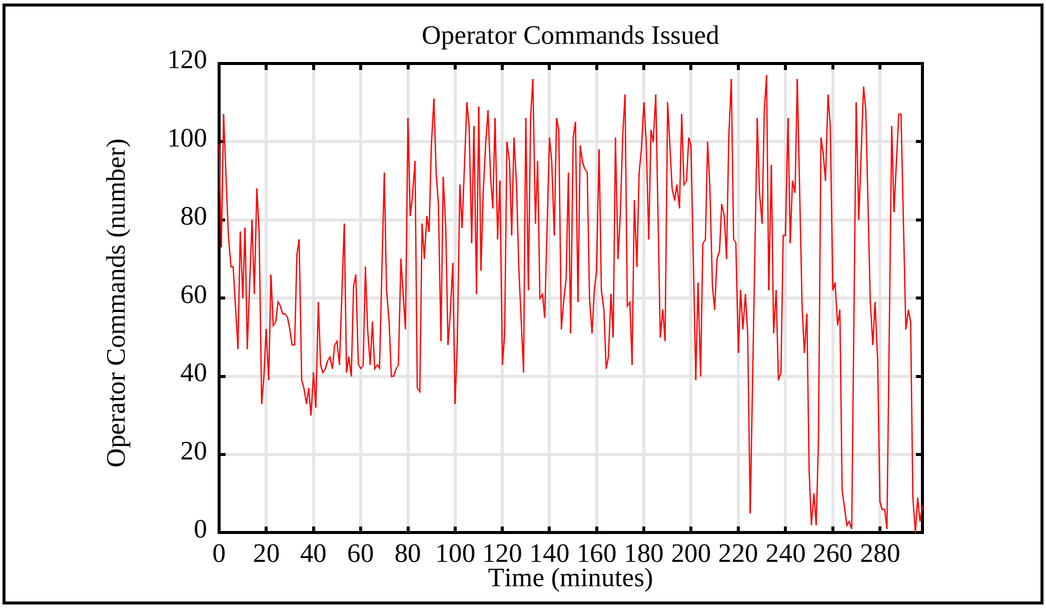
Criteria	Measurement
CPU Usage	dpnProxy (11 seconds) dpnPoller (13 minutes, 46 seconds)
Process Size	dpnProxy (324 kilobytes) dpnPoller (976 kilobytes)

3000 port network

The figure “Commands for 3000 port network” (page 146) shows the number of operator commands issued into the DPN network for a 300 minute period (extended due to lengthened topology and port discovery phases) to support surveillance of 3000 ports.

> ***Note:*** For the purposes of this configuration, the *DPN Agent* was configured for connection to a superior OA one level above that of the previous tests (in order to provide exposure to a sufficient number of network elements to yield the 3000 port loading).

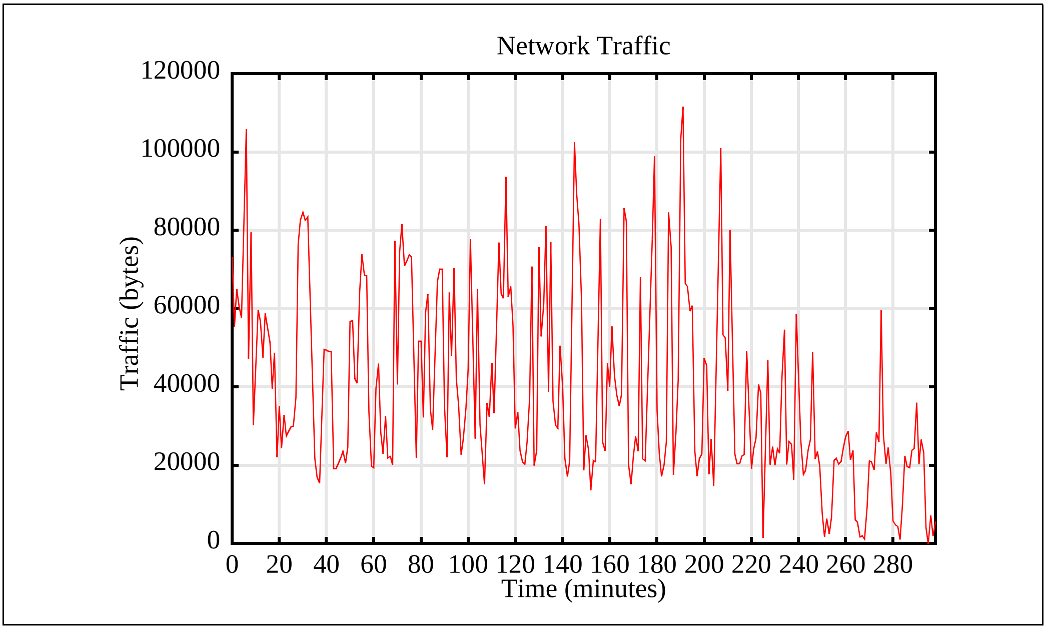
Figure 23
Commands for 3000 port network



In the figure “Commands for 3000 port network” (page 146), the *DPN Agent* concludes the topology and port discovery phases of operation after approximately 250 minutes at which point passive servicing begins with its characteristic spiking on 15 minute boundaries.

The network traffic generated to transmit the operator commands and receive the results during this period is provided in the figure "Traffic for 3000 port network" (page 147).

Figure 24
Traffic for 3000 port network



During topology and port discovery, the *DPN Agent*, on average, generated approximately 50 to 60 kilobytes of network traffic per minute. Upon entering the passive servicing phase of operations, traffic remained at approximately 250 kilobytes per 15 minute interval.

The table “24 hour observations for 3000 ports” (page 148) documents *DPN Agent* process characteristics for a 24 hour operating period.

Table 31
24 hour observations for 3000 ports

Criteria	Measurement
CPU Usage	dpnProxy (11 seconds) dpnPoller (45 minutes, 51 seconds)
Process Size	dpnProxy (324 kilobytes) dpnPoller (1 492 kilobytes)

Overall observations

The following observations of the *DPN Agent* operational characteristics are made based on the four sample configurations:

- the *DPN Agent* conforms to an operational model in which sustained activity is present during network topology and port discovery phases followed by periodic (that is, 15 minute) intervals in which port and module maintenance tasks are performed
- network and port discovery requires approximately 80 minutes per 1000 ports
- CPU utilization is very low and never exceeds 5% of overall processor capability
- *DPN Agent* process sizes relatively small with modest growth for increased number of supported ports

The dynamic behavior of the *DPN Agent* enforces the need to correctly size and instrument the *DPN Agent* for the target surveillance network in order to balance desired operating characteristics such as network discovery time and generated traffic load.

Appendix D Event configuration

This appendix consists of a set of procedures to provide accurate and reliable network status information using *DPN Agent* events.

> ***Note:*** The *DPN Agent* can be used with any SNMP network management application and we have chosen HP OpenView for example purposes, only.

About HP OpenView

The following procedure uses HP OpenView Network Node Manager (NNM) 5.01. This software is not supplied with either the *DPN Agent* package or the Preside Multiservice Data Manager (MDM) package and must be obtained separately.

For more information about the HP OpenView Network Node Manager, see the following HP OpenView documents:

- *Network Node Manager Products Installation Guide*
- *A Guide to Scalability and Distribution for Network Node Manager*
- *Network Node Manager 5.01 Performance and Configuration Guide*
- *Using Network Node Manager*

> ***Note:*** HP OpenView Network Node Manager documents are available on the Web in PostScript and PDF format at http://www.hp.com/openview/index.html.

Configuring DPN Agent events

The following procedures provide instructions to set up HP OpenView to capture and report *DPN Agent* events. These procedures include

- "Installing the MIB" (page 150) describes how to install the *DPN Agent* MIB file DPN-100.mib.
- "Adding an event category" (page 151) describes how to add an event category for the *DPN Agent*.
- "Setting trap formats" (page 151) describes configuration of trap formats for the *dpnTrap* and *dpnAgentEventTrap* events.

Note: These procedures assume that you have installed and configured HP OpenView and have discovered the devices in your DPN network.

Installing the MIB

1 Login to the platform as userID root.

2 Copy the MIB definition file /opt/MagellanNMS/dpnAgent/doc/ \ DPN-100.mib (for example) to an appropriate location on the management system. For example, type the following:

```
cp /opt/MagellanNMS/dpnAgent/doc/DPN-100.mib \
/var/opt/OV/share/snmp_mibs/Vendor/nortel/
```

3 In a UNIX xterm window, start HP OpenView by typing

```
/opt/OV/bin/ovw &
```

The About OpenView window opens (and closes on its own).

The Root map opens, displaying the *Nortel Networks* symbol.

The Event Categories window opens. It may display a status message at the bottom, stating the percentage of the trapd.logs file that has been loaded.

4 From the main menu bar of the Root map, select *Options->Load/Unload MIBs:SNMP*.

5 Click *[Load...]*.

The Load MIB From File dialog window opens.

6 Select the directory location of the DPN-100.mib file. You can also specify the directory location if the directory is not displayed in the provided list.

7 Select file DPN-100.mib from the *Files* selection list. Click *[OK]*.

You have completed the installation of the *DPN Agent* MIB file.

Adding an event category

1 From the main menu bar, select *Options->Event Configuration*.

The *Event Configuration* dialog window opens.

2 From the Event Configuration dialog window, select *Edit->Configure->Event Catgories*.

The *Event Categories* dialog window opens.

3 To add a new category, type **`DPN Events`** in the *Category name* box. Click *[Add]* then click *[Close]*.

All future *DPN Agent* generated events are placed in the new catagory and these events can be viewed in a DPN Events Browser.

You have completed the addition of a new event category for the *DPN Agent*.

Setting trap formats

1 From the Event Configuration dialog window, select *dpnAgentProduct*.

Events are displayed in the *Event Identification* window.

2 From the Event Identification window, select the *dpnTrap* event.

3 If the *DPN Agent* is not configured to generate traps with the alarm code used as the specific trap code, select *Edit->Modify Event* from the menu bar of the Event Configuration dialog window.

The Event Configurator dialog window opens. Go to step 6.

If the *DPN Agent* is configured to generate traps with the alarm code used as the specific trap code, select *Edit->Copy Event* from the menu bar of the Event Configuration dialog window.

The *Event Configurator* dialog window opens.

Note: For more information about configuring alarm codes as specific trap codes, see *mapAlarmCode* in “dpnProxy.cnf Configuration Options” (page 29).

4 From the Event Configurator dialog window, select *Event name* and type a new name; for example **`dpnTraps`**.

5 From the Event Configurator dialog window, select *Event Object Identifier* and replace the `1` at the end of the line with an asterik “*”, which substitutes as a wildcard.

6 From the Event Configurator dialog window, select *Category* and then select *DPN Events*.

7 From the Event Configurator dialog window, select *Event Log Message*.

8 Edit the format of the trap as required. The variables of the trap are defined in “DPN Enterprise MIB” (page 71) and include: dpnTrapType, dpnTrapNotificationId, dpnTrapCompId, dpnTrapCustomerId, dpnTrapReportingOA, dpnTrapFaultArea, dpnTrapFaultCode, dpnTrapFaultMnemonic, dpnTrapProbableCause, dpnTrapSeverity, dpnTrapOperatorData, dpnTrapExpertData, dpnTrapCommentData, and dpnTrapTextual.

The Help dialog provides additional information about HP OpenView variables that can be inserted.

The following line is an example of the trap variables format:

```
$2 DPN “ $3 “ Type:(1=msg,2=set,3=clr) $1 Sev: $10 \
Code: $7 Mnemonic; $8 Comment: $13
```

The $2, $3, $1, $10, $7, $8, and $13 are place holders for the actual variables from the trap. Use the list of trap variables above for substitution. Literal strings do not require quotes.

9 Click *OK*.

10 From the Event Configurator dialog window, select *dpnAgentEventTrap*.

11 Edit the format of the trap as required. The variables of the trap are defined in “DPN Enterprise MIB” (page 71) and include: dpnAgentEventType, dpnTrapCompId and dpnTrapTextual.

The Help dialog provides additional information about HP OpenView variables that can be inserted.

The following line is an example of the trap variables format:

```
(1=start,2=end,3=discard mib full,4=discard
un-supported,5=parse error) $1 compId $2 $3
```

The $1, $2, and $3 are place holders for the actual variables from the trap. Use the list of trap variables above. Literal strings do not require quotes.

12 Click *OK*.

13 From the main menu bar of the Event Configuration dialog window, select *File->Save*.

You have completed the configuration of trap formats for the *dpnTrap* and *dpnAgentEventTrap* events

Index

Preside Multiservice Data Manager

DPN SNMP Agent

User Guide

Release: R14.3

Publication: 241-6001-210
Document status: Standard
Document version: 14.3RSUP
Document date: December 2003
Printed in Canada