Print Test Book

Edition 3

HP 9000 Networking

Manufacturing Part Number: B5139-90023
E0601

United States

Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett- Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

HEWLETT-PACKARD COMPANY
3000 Hanover Street
Palo Alto, California 94304
U.S.A.

Use of this manual and flexible disk(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

California.

Contents

1. Introduction to the NOF API

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Preface

HP-UX SNAplus2 NOF Programmers Guide contains the information required to develop C-language application programs that use the Node Operator Facility (NOF) API to manage SNAplus2 resources. SNAplus2 is a software product that enables a server running HP-UX to exchange information with other nodes on an SNA network.

The guide provides detailed reference information for experienced NOF programmers.

This guide applies to Release 6 of SNAplus2.

Prerequisite Knowledge

Before reading this manual, you should have a knowledge of the following subjects:

- SNA and APPN concepts (see “Related Publications”)
- The C compiler cc
- HP-UX version 10.10 or later

Organization of This Book

This book is organized as follows:

Chapter 1, “Introduction to the NOF API.”

Provides an overview of the SNAplus2 NOF API and the functions it provides.

Chapter 2, “Writing NOF Applications.”

Contains general information a programmer needs when writing NOF applications and information about compiling and linking the applications.

Chapter 3, “NOF API Verbs (ACTIVATE_SESSION to OPEN_FILE).”

Provides a detailed description of the NOF verbs, including parameters and return codes. The verb descriptions are arranged alphabetically and divided over chapters 3, 4, and 5.

Chapter 4, “NOF API Verbs (QUERY Verbs).”

Provides a detailed description of the NOF QUERY verbs, including parameters and return codes.

Chapter 5, “NOF API Verbs (REGISTER_INDICATION_SINK to UNREGISTER_INDICATION_SINK).”

Provides a detailed description of the NOF verbs from REGISTER_INDICATION_SINK to UNREGISTER_INDICATION_SINK, including parameters and return codes.

Chapter 6, “NOF Indications.”

Provides a detailed description of each of the indications that a NOF application can register to receive.

Appendix A, “Common Return Codes.”

Provides information about return codes that are common to all the NOF verbs.

Typographic Conventions

Table 1, “Typographic Conventions,” shows the typographic styles used in this document.

Table 1 Typographic Conventions

Special Element	Sample of Typography
Document title	*HP-UX SNAplus2 NOF Programmers Guide*
File or path name	`sna.net`
Directory name	`/var`
Header file	`nof_c.h`
Program or application	`snapadmin`
Command	`define_local_lu`; `cd`
General reference to all verbs of a particular type	DEFINE_* (indicates all of the NOF API verbs that define resources)
Option or flag	`-I`
Parameter	*opcode*
Literal value or selection that the user can enter (including default values)	`255`
Constant	`AP_READ_ONLY`
Return value	`AP_INVALID_FORMAT`; `0`
Variable representing a supplied value	*a.b.c.d*
Environment variable	`LD_RUN_PATH`
Programming verb	CONNECT_NODE
User input	**`cc -I /opt/sna/include -L /opt/sna/lib -lnof -lsna`**

Table 1 **Typographic Conventions**

Special Element	Sample of Typography
Function, call, or entry point	`ioctl`
Data structure	`NOF_CALLBACK`
Hexadecimal value	`0x20`

SNAplus2 Publications

SNAplus2 publications include user guides, administrator guides, and programmer guides. The following sections describe the contents of each book.

Publications for Users

SNAplus2 provides the following user's guides:

HP-UX SNAplus2 General Information

Provides an introduction to SNAplus2 and explains key product concepts and features.

HP-UX SNAplus2 3270/3179G Users Guide

Explains how to perform the following functions when you use 3270 emulation:

- Starting and stopping 3270 emulation
- Transferring files
- Using customization features such as remapping your keyboard and displaying colors
- Interpreting status-line information
- Sending NetView user alerts
- Viewing response times

HP-UX SNAplus2 RJE Users Guide

Explains how to start and stop the RJE workstation, queue a job for submission to the host, list the queued jobs, cancel a queued job, and send commands to the host's job entry subsystem (JES) console.

HP-UX SNAplus2 and TN3270 Glossary

Provides a comprehensive list of terms and their definitions used in the SNAplus2 library.

Publications for Administrators

SNAplus2 provides the following administrator guides:

HP-UX SNAplus2 Installation Guide

Explains how to install the SNAplus2 software and set up system files.

HP-UX SNAplus2 Upgrade Guide

Provides information about upgrading to the current version of SNAplus2 from previous versions. It includes information about converting configuration files, rebuilding applications that use the SNAplus2 application program interfaces (APIs), and changes in other SNAplus2 functions.

HP-UX SNAplus2 Administration Guide

Explains how to enable, configure, and manage SNAplus2. This guide provides information about SNA concepts, and an overview of the features provided by SNAplus2. It describes how to configure and manage SNAplus2 using the Motif administration program and provides guidance for users of the SNAplus2 command-line administration program.

HP-UX SNAplus2 Administration Command Reference

Explains how to use the SNAplus2 command-line administration program and shows the syntax of all SNAplus2 administration commands.

HP-UX SNAplus2 Diagnostics Guide

Explains how to investigate and resolve common problems and provides an overview of diagnostic tools, including logging and tracing.

Publications for Programmers

SNAplus2 provides the following programmer's guides. Each guide includes conceptual and detailed reference information.

HP-UX SNAplus2 APPC Programmers Guide

Contains the information you need to write application programs using Advanced Program-to-Program Communication (APPC).

HP-UX SNAplus2 CPI-C Programmers Guide

Contains the information you need to write application programs using Common Programming Interface for

Communications (CPI-C).

HP-UX SNAplus2 3270 & TN3270 HLLAPI Programmers Guide

Contains the information you need to write application programs using High-Level Language Application Program Interface (HLLAPI).

HP-UX SNAplus2 LUA Programmers Guide

Contains the information you need to write applications using the Conventional LU Application Programming Interface (LUA).

HP-UX SNAplus2 CSV Programmers Guide

Contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).

HP-UX SNAplus2 MS Programmers Guide

Contains the information you need to write applications using the Management Services (MS) API.

HP-UX SNAplus2 NOF Programmers Guide (this guide)

Contains the information you need to write applications using the Node Operator Facility (NOF) API.

Related Publications

For information about SNA, APPN, or LU 6.2 architecture, refer to the following IBM documents:

- *Systems Network Architecture:*
  - — *APPN Architecture Reference*, SC30-3422
  - — *Format and Protocol Reference Manual: Architectural Logic*, SC30-3112
  - — *Formats*, GA27-3136
  - — *LU6.2 Reference: Peer Protocols*, SC31-6808
  - — *Management Services Reference*, SC30-3346
  - — *Sessions between Logical Units*, GC20-1868
  - — *Technical Overview*, GC30-3073
- *APPN Architecture and Product Implementations Tutorial*, GG24-3669
- *AS/400 Advanced Peer-to-Peer Networking*, GG24-3287
- *System/370 Principles of Operation*, GA22-7000

1 Introduction to the NOF API

Overview

This chapter provides an introduction to the SNAplus2 NOF API. It includes the following information:

- Purpose of the NOF API
- SNAplus2 components and resources
- Client-server operation
- NOF verbs and indications

Purpose of the NOF API

The SNAplus2 NOF API provides access to a standard set of commands, called NOF verbs, that can be used to administer the SNAplus2 system from within an application program. These verbs enable you to define and delete resources, specify SNAplus2 parameters such as diagnostics levels and file names, start and stop defined resources, query the definition or current status of resources, and manage which servers on the SNAplus2 LAN can act as backup masters if the master configuration file server is not available.

The NOF verbs provide the same functions as commands issued to the command-line administration program `snapadmin`, or as records in a SNAplus2 configuration file. For example, the NOF verb DEFINE_LOCAL_LU is equivalent both to a `define_local_lu` command issued to the `snapadmin` program, and to a `define_local_lu` record in a configuration file; all three of them perform the same function, which is to specify the parameters of a SNAplus2 local APPC LU.

You can use the Motif administration program `xsnapadmin` to perform the same function as a NOF verb or an administration command (for example, you can use the program to define a local APPC LU). However, this program does not provide access to the full range of parameters included in some NOF verbs. For more information about using the Motif administration program, refer to the *HP-UX SNAplus2 Administration Guide*.

You can issue NOF verbs to any of the following targets:

- A running SNAplus2 node (to manage its resources or to monitor its operation)
- A server where the node is not running (to query the stored configuration or to modify it for use when the node is next started)
- The SNAplus2 domain as a whole-to define, modify, or query the configuration of domain resources (resources used to support particular user programs, such as 3270 diagnostics parameters and CPI-C side information entries, which are not associated with a particular node).

The NOF API enables you to do the following:

- Develop your own application programs to manage the SNAplus2 system
- Develop application programs that use the other SNAplus2 APIs so that they can also manage their own resources (for example, an APPC application can check that the communications link to its partner TP is active before attempting to allocate a conversation or can define the remote LU where its partner TP is located).

Node Configuration File

Configuration information for each SNAplus2 node is held in a text file on the computer where the node runs. This file includes information about the node's resources, and specifies which resources will be active when SNAplus2 is started. When you start the node, the file provides an initial definition of the resources that are available; you can then use the NOF API or the SNAplus2 administration tools to modify the running node's resources as your requirements change.

You can set up multiple configuration files, to store different SNAplus2 configurations for use at different times, and select which of these files to use when starting the SNAplus2 software.

Configuration in an APPN network is a dynamic process; you can add, delete, or modify resources as necessary while the SNAplus2 software is running. The configuration file provides an initial definition of the available resources and stores the current definition so that you can use it again when you need to restart the node, but it is not necessary to define the entire configuration before starting the SNAplus2 software.

Domain Configuration File

Configuration information for SNAplus2 domain resources is held in a single text file on the master server. You can set up multiple domain configuration files, to store different SNAplus2 configurations for use at different times, and select which of these files to use when starting the SNAplus2 software on the master server.

Configuration in an APPN network is a dynamic process; you can add, delete, or modify resources as necessary while the system is running. The domain configuration file provides an initial definition of the available domain resources and enables you to store the current definition so that you can use it again when you need to restart the system, but it is not necessary to define the entire domain configuration before starting the

SNAplus2 software or to restart the software when you make changes.

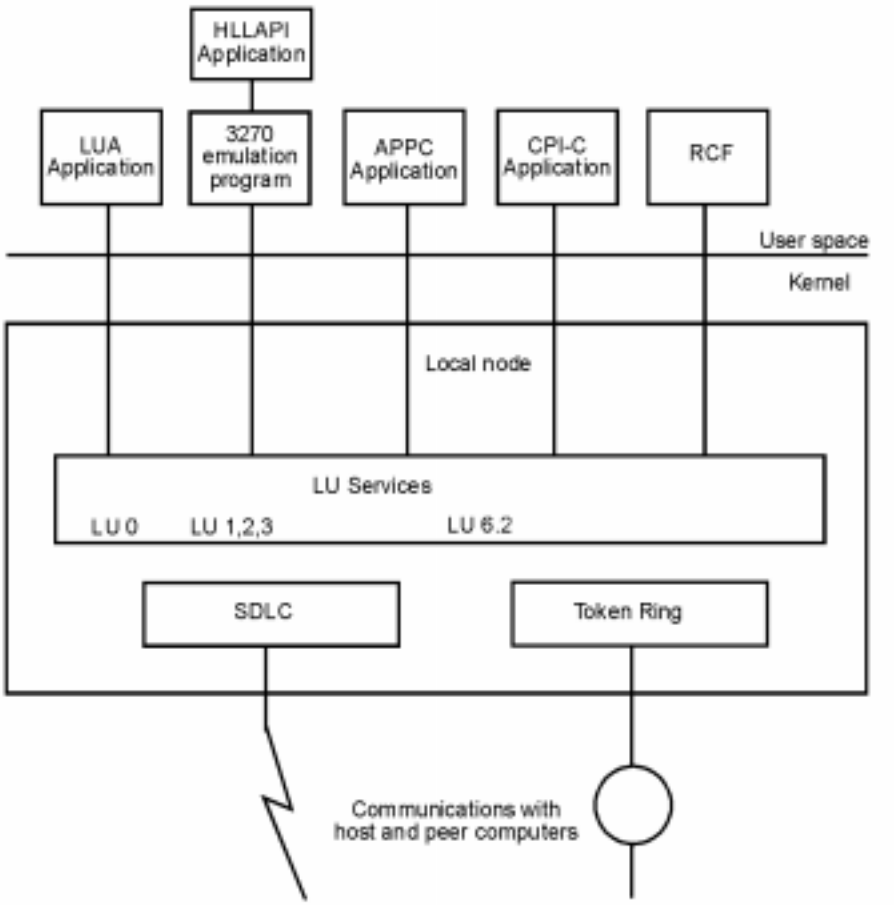
SNAplus2 Components

SNAplus2 implements an APPN node to communicate with other nodes on the SNA network. This provides logical unit (LU) 6.2 support for APPC and CPI-C capabilities and for 5250 emulation, as well as LU 0, 1, 2, and 3 support for 3270, RJE, and LUA communications.

SNAplus2 can operate either as a LEN node or as an APPN end node, depending on its configuration. Certain functions are supported only on end nodes, as defined by the APPN architecture. These differences are indicated where necessary in this manual; where no differences are indicated, the information applies to both node types.

Figure 1-1, “SNAplus2 Components,” shows the components of SNAplus2 and how they work together.

Figure 1-1 SNAplus2 Components



The local node, including its associated connectivity resources (DLCs, ports and LSs), is implemented as STREAMS components in the kernel of the HP-UX system.

The 3270 emulation program, RJE workstation, APPC transaction programs, CPI-C applications, LUA applications, and the Remote Command Facility (RCF) are user-space programs.SNAplus2 supports multiple copies of the 3270 emulation program, and multiple APPC TPs, CPI-C applications, and LUA applications, running concurrently.

SNAplus2 Resources

The resources of the SNAplus2 system can be divided into two types:

Node resources

Define the communications capabilities of a particular APPN node:

- Connectivity resources:
  - — DLCs
  - — Ports
  - — Link stations
  - — Connection networks
- LUs
  - — Type 0-3 for:
    - — 3270
    - — RJE
    - — LUA communications
  - — Type 6.2 for:
    - — APPC
    - — CPI-C communications
    - — 5250 emulation
- Modes and classes of service
- Directory information

Domain resources

Additional resources not defined as part of the node, which are used to support particular user programs. These resources include:

- 3270 user information
- RJE workstations
- CPI-C side information
- Information about access to the HP-UX Command

Facility and Service Point Command Facility

For more information about these resources, refer to the *HP-UX SNAplus2 Administration Guide*.

Dependent LU Requester (DLUR)

This section does not apply to LEN nodes.

As well as providing direct access to a host computer, SNAplus2 can also provide DLUR facilities. This feature enables sessions for dependent LUs to span multiple nodes in an APPN network, instead of requiring a direct connection to the host.

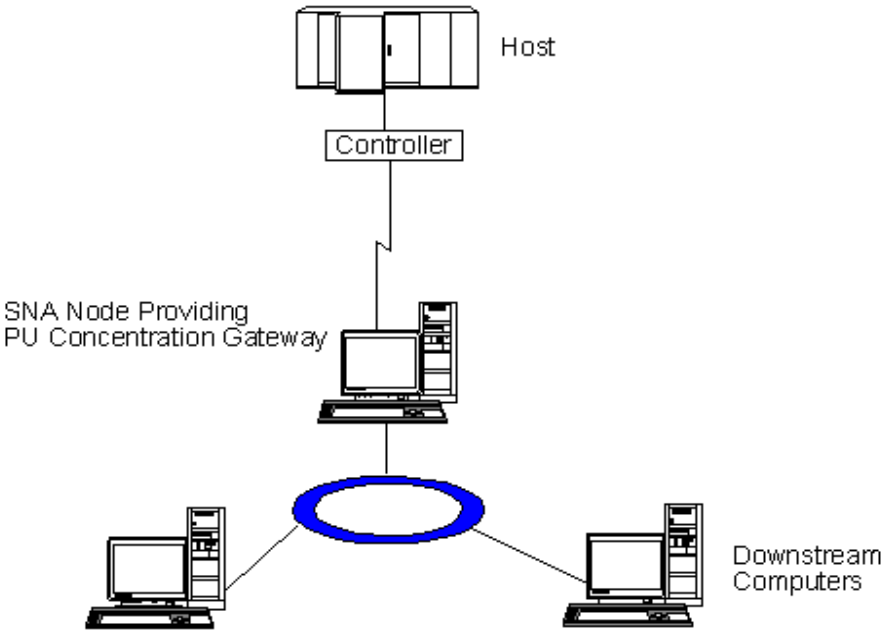
DLUR on the SNAplus2 node works in conjunction with Dependent LU Server (DLUS) at the host to route sessions from dependent LUs on the SNAplus2 node across the network to the DLUS host. The route to the host can span multiple nodes, and can take advantage of APPN's network management, dynamic resource location, and route calculation facilities.

PU Concentration

As well as providing direct access to a host computer, SNAplus2 can also provide PU concentration facilities. This feature enables other computers to use SNAplus2 resources to access a host computer instead of requiring a separate connection to the host from each computer.

Figure 1-2 illustrates the PU concentration feature.

Figure 1-2 PU Concentration Facilities Provided by SNAplus2



The downstream computer must contain an SNA PU type 2.0 or 2.1 to support dependent LUs. For example, the downstream computer could be a PC running Microsoft's SNA Server for Windows NT or another SNAplus2 computer.

Using this feature, all the data transferred between the host and the downstream computer is routed through the SNAplus2 local node. This enables a downstream computer to share a host connection with SNAplus2 or with other downstream computers instead of requiring a direct link. For example, you could set up several downstream computers connected to SNAplus2 over a local Token Ring network, so that they could all access the same long-distance leased line from SNAplus2 to the host.

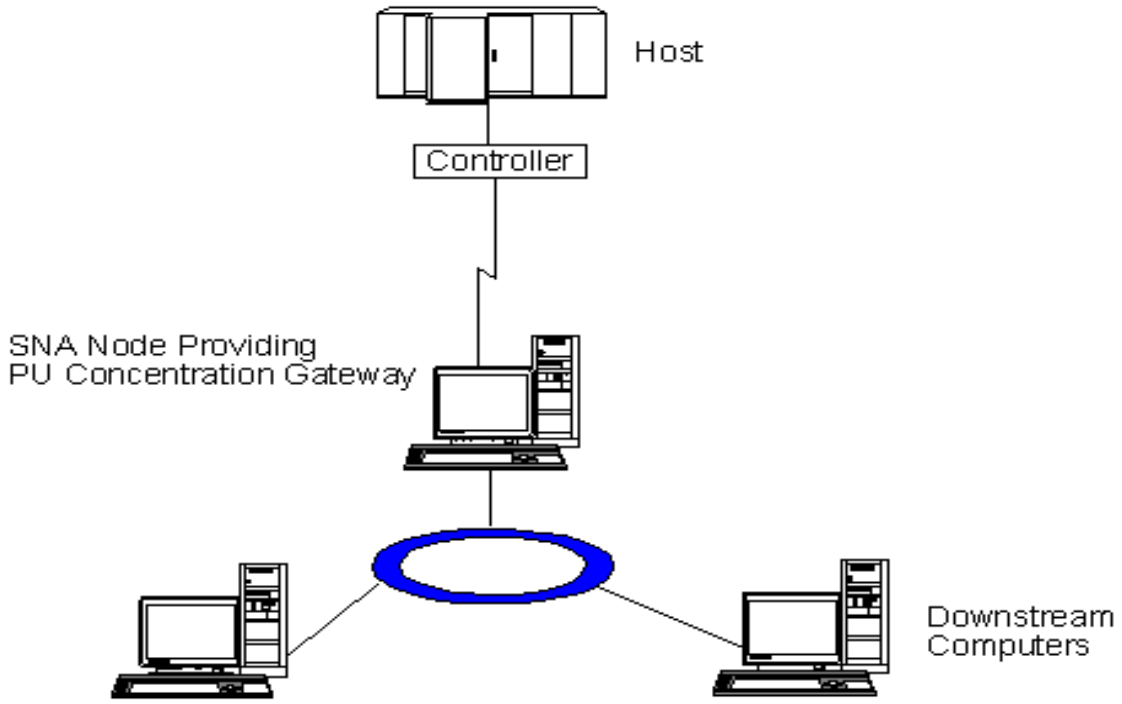
Using PU concentration also simplifies the configuration at the host because there is no need to define the downstream computers and the communications links to them. The host configuration only needs to include the SNAplus2 computer and its host communications link; the LUs at the downstream computers are configured as part of the resources of the SNAplus2 computer. The host computer is not aware that PU concentration is being used.

TN Server

SNAplus2 TN server provides access to 3270 host computers for TN3270 users on other computers. The TN server feature enables TN3270 users to share a host connection with SNAplus2 or with other TN3270 users, instead of requiring a direct link. It also enables TN3270 users to access hosts that are not running TCP/IP.

Figure 1-3, “SNAplus2 TN Server,” illustrates the SNAplus2 TN server function:

Figure 1-3 SNAplus2 TN Server



The SNAplus2 TN server feature provides an association between a TN3270 user and SNAplus2 3270 LU. All data from the TN3270 user is routed to the LU. This means that the configuration for both the host and the TN3270 user is as though they were connected directly; neither needs to be aware that data is being routed through TN server.

TN Server Users

Any 3270 emulation programs that communicate over TCP/IP rather than over an SNA network are referred to as TN3270 programs (Telnet 3270 emulation programs). SNAplus2 TN server supports various TN3270 programs, including SNAplus2 TN3270 (packaged separately from the main SNAplus2 product).

When a TN3270 program communicates with TN server, SNAplus2 identifies it by the TCP/IP address of the computer where the TN3270 program is running; it cannot distinguish between two different TN3270 programs being used by different users on the same computer. In the SNAplus2 manuals, the term “TN server user” refers to the computer where a TN3270 program is running (identified by its TCP/IP address), and not to an individual user of that program.

Each TN server user is normally configured to access a single 3270 LU and so is restricted to one host session at a time. However, you can also configure a TN server user to access a pool of 3270 LUs instead of having a single dedicated 3270 LU for each user. This enables the user to access as many sessions as there are available LUs in the pool.

Client-Server Operation

The computers on the SNAplus2 LAN are of two types: servers and clients. A server contains a SNAplus2 node and its associated connectivity components; a client does not contain these connectivity components, but accesses them on the server by means of the LAN. Servers must be HP-UX computers; clients may be running HP-UX or Windows systems. Servers and clients communicate across the LAN using Berkeley Software Distribution (BSD) Sockets.

Each SNAplus2 LAN system, referred to as a domain, is identified by a domain name. This name is specified during the installation of each SNAplus2 computer (server or client), so that all computers in a single SNAplus2 LAN system have the same domain name. To install two separate SNAplus2 domains on the same physical network, you simply use two different domain names to identify which domain each computer belongs. A single SNAplus2 domain can correspond to a TCP/IP subnet, can be part of a TCP/IP subnet (so that there are two or more separate SNAplus2 domains in the same subnet), or can span multiple subnets.

Each server maintains information about its own node configuration in a node configuration file. You can use the SNAplus2 administration tools or the NOF API to examine and modify the node's configuration. This can be done either from this server or from any other computer in the domain, as long as the SNA software is running (whether or not the node is started).

Information about the configuration of domain resources for the complete SNAplus2 LAN is held in a domain configuration file. If you have more than one server on the LAN, SNAplus2 ensures that this information is consistent across all servers.

Master Server and Backup Servers

If you are using SNAplus2 with all programs on one computer or on a LAN that contains only one server, you do not need to read this section.

At any time, one server on the LAN, known as the master server, holds the master copy of the SNAplus2 domain configuration file. You can define other servers on the LAN to be backup servers; the domain configuration file is copied to backup servers (either when they are started or when the master copy is changed) so that all backup servers

hold a copy of the latest information.

If the master server fails or if the SNA software on that computer is stopped, a backup server takes over as the master. The domain configuration file on this server is used as the master copy and is copied to other servers as necessary. When the master server is restarted, it receives a copy of the domain configuration from the backup server currently acting as master and then takes over as the master.

In general, define at least one backup server in addition to the master server. Any remaining servers can be defined as additional backup master servers or they can be left as peer servers. A peer server obtains configuration information from the master server as required but cannot act as a backup server.

If at any time the master server and all backup servers are inactive, a node on a peer server can still operate, and you can still change the node's configuration. However, you cannot access the domain configuration file and therefore cannot access the configuration of domain resources (as opposed to node resources). This means that you will not be able to start the 3270 emulation program or RJE programsor allocate CPI-C conversations using symbolic destination names defined in the configuration file.

There is one situation in which SNAplus2 cannot maintain a consistent configuration of domain resources across the LAN; it is your responsibility to maintain the configuration in this case. This situation occurs when the LAN is split by a network failure into two noncommunicating domains, each containing one or more backup servers. In this situation, there will be an acting master server in each domain, which will hold any changes made to the domain configuration file in that domain but will be unaware of any changes made in the other domain. When the LAN connection is re-established, the domain configuration file from the original master server (or from the highest backup server available in either of the two domains if the master is inactive at this point) will become the domain configuration file across the LAN; this will overwrite any changes made to the domain configuration file in the other domain while the network was split. Because of this, do not attempt to make any changes to the domain configuration file in either of the two domains while the LAN connection is broken. Changes can be made to the configuration of individual nodes.

SNAplus2 stores information about the master server and backup servers in the file `sna.net`, known as the SNA network data file. The master copy of this file is stored on the master server; any changes made

to it are automatically copied to all other servers, in the same way that changes to the domain configuration file are copied to backup servers. You cannot edit the contents of the file directly; instead, SNAplus2 provides NOF verbs to access the file.

For more information about the SNA network data file, refer to the *HP-UX SNAplus2 Administration Command Reference*.

HP-UX Clients

A client computer does not contain any configuration file or the SNA network data file; it holds only the information it needs to access servers on the SNAplus2 LAN and relies on a server to provide the necessary configuration information.

The SNA network information required is held in the file `/etc/opt/sna/sna_clnt.net` . For more information about this file, refer to the *HP-UX SNAplus2 Administration Command Reference*.

Windows Clients

The SNA network information, and other information required by Windows clients, is held in the file `sna.ini`. For more information about this file and on managing Windows clients, refer to the *HP-UX SNAplus2 Administration Guide*.

NOF Verbs to Manage Specific SNAplus2 Functions

The following sections list the NOF verbs that are relevant to particular SNAplus2 functions. For more information about individual verbs, see Chapters 3, 4, and 5.

Managing the Target (Node or File) for NOF Verbs

A NOF verb can be issued to a node, to the domain configuration file, or to the SNA network data file. To access the target node or file, use one of the following verbs:

- OPEN_FILE
- CONNECT_NODE

When you issue the verbs shown above to access the target, you are initially restricted to issuing verbs that query the configuration; you cannot issue verbs to modify it. To obtain write access to the target node or file so that you can issue verbs that modify the configuration, use the following verb:

- SET_PROCESSING_MODE

To register for indications when the target configuration changes, use the following verb:

- REGISTER_INDICATION_SINK

To unregister when indications are no longer required, use the following verb:

- UNREGISTER_INDICATION_SINK

To release the target node or file when you have finished issuing NOF verbs, use one of the following verbs:

- DISCONNECT_NODE, CLOSE_FILE

The CONNECT_NODE and DISCONNECT_NODE verbs can be issued on the HP-UX client computers as well as on a server; this enables you to use SET_CS_TRACE, QUERY_CS_TRACE, SET_TRACE_FILE, and QUERY_TRACE_FILE to control client-server tracing for the client. For

more information, see “SET_CS_TRACE”, “QUERY_CS_TRACE”, “SET_TRACE_FILE”, and “QUERY_TRACE_FILE”. No other NOF verbs can be issued to client computers.

Getting Started

The first step is to define the SNAplus2 node that runs on each computer, and its communications links to other computers. To define these components, use the following verbs:

- DEFINE_NODE
- DEFINE_DLC, DEFINE_PORT, DEFINE_LS

After defining these components, activate them to establish the link to the remote system. (DLCs, ports, and LSs can be defined to be “initially active” using the DEFINE_* verbs described previously, so that they are started automatically when the node is started; in this case, it is not necessary to start them manually.) To activate components, use the following verbs:

- INIT_NODE
- START_DLC, START_PORT, START_LS

The components must be started in the order shown because each component relies on the one before it.

To stop these components when access to the remote system is no longer required, use the following verbs:

- STOP_LS, STOP_PORT, STOP_DLC

To obtain information about the configuration or current status of these components, use the following verbs:

- QUERY_NODE
- QUERY_DLC, QUERY_PORT, QUERY_LS

To obtain information about the usage of an LS, use the following verb:

- QUERY_STATISTICS

To delete connectivity components when they are no longer required, use the following verbs:

- DELETE_DLC, DELETE_PORT, DELETE_LS

If you are communicating with many nodes on the same shared-access transport facility (SATF), you can set up a connection network (CN) to represent these nodes, instead of having to define explicit LSs to each node. CNs cannot be used if the local node is a LEN node.

To set up the CN, you first define a DLC and port to access each of the nodes on the SATF.

You then define a CN that includes all these ports; you do not need to define any LSs because a dynamic LS to the CN will be set up as required. To define the CN, or to add ports to an existing CN, use the following verb:

- DEFINE_CN

To obtain information about defined CNs, or about the ports on a CN, use the following verbs:

- QUERY_CN, QUERY_CN_PORT

To delete a CN when it is no longer required, or to remove ports from a CN without deleting the CN, use the following verb:

- DELETE_CN

To stop the node, which deactivates all resources associated with it, use the following verb:

- TERM_NODE

To define default parameters used by the node, or to query the definition of these parameters, use the following verbs:

- DEFINE_DEFAULTS, QUERY_DEFAULTS

To query the options and limits permitted by your SNAplus2 license for the node, use the following verb:

- QUERY_NODE_LIMITS

3270 Communications

If SNAplus2 users will be using 3270 emulation to communicate with host systems, you need to define the communications link to the host. For more information, see “Getting Started”. The definition of the LS to the host must include the name of a local PU to own the LUs required for 3270 emulation and must have the *solicit_sscp_sessions* parameter set to AP_YES.

You then need to define LUs that can be used for 3270 emulation and also the users that can access these LUs. To define LUs and users, use the following verbs:

- DEFINE_LU_0_TO_3, DEFINE_LU_0_TO_3_RANGE
- DEFINE_EMULATOR_USER

To obtain information about the configuration or current status of LUs, use the following verb:

- QUERY_LU_0_TO_3

To obtain information about the PU that owns an LU, use the following verb:

- QUERY_PU

To obtain information about the definition of a 3270 user, about the user's current 3270 emulation program usage, or about individual sessions being accessed by the user, use the following verbs:

- QUERY_EMULATOR_USER_DEF, QUERY_3270_USER, QUERY_3270_USER_SESSIONS

To delete LUs when they are no longer required or to delete users so that they can no longer access 3270 emulation, use the following verbs:

- DELETE_LU_0_TO_3, DELETE_LU_0_TO_3_RANGE
- DELETE_EMULATOR_USER

If you want to provide LU pools (groups of LUs that can be assigned to user sessions as required, rather than having an LU explicitly defined for each user session), use the following verbs to define a pool, to obtain information about the definition, or to delete a pool or remove LUs from it when no longer required:

- DEFINE_LU_POOL, QUERY_LU_POOL, DELETE_LU_POOL

If you want to enable 3270 users to send alert messages to the host NetView operator or if you want to specify how the 3270 emulation program should classify response-time information that users can view with the Response Time Monitor feature, use the following verbs to define alerts and response-time boundaries or to obtain information about the current definitions:

- DEFINE_3270_DIAG, QUERY_3270_DIAG

5250 Communications

If SNAplus2 users will be using 5250 emulation to communicate with AS/400 systems, you need to define the communications link to the AS/400. For more information, see “Getting Started”.

You then need to define local LUs that can be used for 5250 emulation. To define LUs, use the following verb:

- DEFINE_LOCAL_LU

If the 5250 emulation program identifies a partner LU by its LU alias, you need to define the partner LU to associate the alias with an LU name. (If the program identifies a partner LU by its LU name, this step is not required.) To define partner LUs, use the following verb:

- DEFINE_PARTNER_LU

To obtain information about the configuration or current status of local LUs, the current status of partner LUs, or the definition of partner LUs, use the following verbs:

- QUERY_LOCAL_LU
- QUERY_PARTNER_LU, QUERY_PARTNER_LU_DEFINITION

To delete LUs when they are no longer required, use the following verb:

- DELETE_LOCAL_LU, DELETE_PARTNER_LU

RJE Communications

If SNAplus2 users will be using RJE to submit jobs to host systems for processing, you need to define the communications link to the host. For more information, see “Getting Started”. The definition of the LS to the host must include the name of a local PU to own the LUs required for RJE and must have the *solicit_sscp_sessions* parameter set to AP_YES.

You then need to define LUs that can be used for RJE and the RJE workstations that can access these LUs.

To define LUs, use the following verbs:

- DEFINE_LU_0_to_3, DEFINE_LU_0_TO_3_RANGE

To obtain information about the configuration or current status of LUs, use the following verb:

- QUERY_LU_0_TO_3

To obtain information about the PU that owns an LU, use the following verb:

- QUERY_PU

To define workstations, use the following verb:

- DEFINE_RJE_WKSTN

To obtain information about the definition of a workstation or about its current status, use the following verbs:

- QUERY_RJE_WKSTN_DEF, QUERY_RJE_WKSTN

To delete LUs or workstations when they are no longer required, use the following verbs:

- DELETE_LU_0_TO_3, DELETE_LU_0_TO_3_RANGE
- DELETE_RJE_WKSTN

LUA Communications

If applications running on SNAplus2 will be using LUA to communicate with host programs, you need to define the communications link to the host. For more information, see “Getting Started”. The definition of the LS to the host must include the name of a local PU to own the LUA LUs, and must have the *solicit_sscp_sessions* parameter set to AP_YES.

You then need to define LUs that can be used for LUA. To define the LUs, use the following verbs:

- DEFINE_LU_0_TO_3 to define an individual LU or DEFINE_LU_0_TO_3_RANGE to define multiple LUs with a single verb

To delete LUs when they are no longer required, use the following verbs:

- DELETE_LU_0_TO_3 to delete an individual LU or DELETE_LU_0_TO_3_RANGE to delete multiple LUs with a single verb

To obtain information about the configuration or current status of LUs, use the following verb:

- QUERY_LU_0_TO_3

To obtain information about the PU that owns an LU, use the following verb:

- QUERY_PU

If you want to provide LU pools (groups of LUs that can be assigned to applications as required, rather than having LUs explicitly defined for each application), use the following verbs to define a pool, to obtain information about the definition, or to delete a pool or remove LUs from it when no longer required:

- DEFINE_LU_POOL, QUERY_LU_POOL, DELETE_LU_POOL

APPC Communications

If applications running on SNAplus2 will be using APPC to communicate with applications running on host or peer computers, you need to define LUs that can be used for APPC.

APPC configuration in an APPN network is much simpler than in a pre-APPN SNA network. Many of the required components, and the interactions between them, can be defined or determined dynamically when sessions are started and do not need to be specified explicitly in the initial configuration.

Each node includes a default APPC local LU (the “control point LU”). An APPC application can use this LU, or you can define additional LUs so that different applications can use different LUs. To define the LUs, use the following verb:

- DEFINE_LOCAL_LU

To obtain information about the configuration or current status of LUs, including the control point LU, use the following verb:

- QUERY_LOCAL_LU

Because APPN can locate a partner LU dynamically when a local application needs to start a session to it, normally you do not need to define partner LUs. However, you may need to define partner LUs if you need to enforce the use of particular APPC features such as conversation security. To define a partner LU, use the following verb:

- DEFINE_PARTNER_LU

To obtain information about the current status of a partner LU or about its definition if it was explicitly defined, use the following verbs:

- QUERY_PARTNER_LU, QUERY_PARTNER_LU_DEFINITION

If the local application communicates with its partner using one of the standard SNA-defined modes, you do not need to define a mode. However, you may want to define additional modes for applications that have particular requirements not covered by the standard modes. To define a mode, use the following verb:

- DEFINE_MODE

To define or query the default mode, which specifies parameters that will be used for any unrecognized mode name, use the following verbs:

- DEFINE_DEFAULTS, QUERY_DEFAULTS

The class of service (COS) used for a mode is normally one of the standard SNA-defined classes of service. However, the node can be configured to support mapping each mode to a specific COS (the *mode_to_cos_map_supp* parameter on the DEFINE_NODE verb). In this case, you may want to define additional COSs for applications that have particular requirements not covered by the standard COSs. To define a COS, use the following verb:

- DEFINE_COS

To specify the default COS to which any unrecognized modes will be mapped, use the following verb:

- DEFINE_MODE

To obtain information about the definition or current usage of a mode, about the COS used by a mode, or about the definition of a COS, use the following verbs:

- QUERY_MODE_DEFINITION, QUERY_MODE, QUERY_MODE_TO_COS_MAPPING
- QUERY_COS, QUERY_COS_NODE_ROW, QUERY_COS_TG_ROW

If the local and partner LUs use session-level security, you need to define the password used to establish a session between the local LU and partner LU. To define the password, check the current definition, or delete the password when it is no longer required, use the following verbs:

- DEFINE_LU_LU_PASSWORD, QUERY_LU_LU_PASSWORD, DELETE_LU_LU_PASSWORD

To delete local LUs, partner LUs, modes, or COSs when they are no longer required, use the following verbs:

- DELETE_LOCAL_LU, DELETE_PARTNER_LU
- DELETE_MODE, DELETE_COS

SNAplus2 negotiates session limits with the partner LU automatically when sessions are established. If you need to manage session limits between a local LU and its partner LU explicitly, use the following verbs:

- INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, RESET_SESSION_LIMIT

To manage individual sessions, use the following verbs:

- QUERY_SESSION
- ACTIVATE_SESSION, DEACTIVATE_SESSION, DEACTIVATE_CONV_GROUP

Normally you do not need to define SNAplus2 invokable TPs if they are operator-started. If a TP is to be automatically started by SNAplus2 when a remote TP allocates a conversation to it, if it is to be operator-started and a broadcast queued TP (which means that incoming conversation requests can be routed dynamically to the TP wherever it is running), or if it is to be operator-started and requires a specific Receive_Allocate timeout value, you need to specify it in the SNAplus2 invokable TP data file. For more information about this file, refer to the *HP-UX SNAplus2 Administration Guide*.

In addition, if a TP (either operator-started or auto-started) needs to be restricted to particular values for conversation security, confirm synchronization, or conversation type (mapped or basic), or if you need to limit the number of instances of the TP that can be running at any time, you need to define the TP. Use the following verb:

- DEFINE_TP

To obtain information about the definition of a TP, about its current usage, or about currently active invokable TPs, use the following verbs:

- QUERY_TP_DEFINITION, QUERY_TP, QUERY_AVAILABLE_TP

To delete a defined TP when it is no longer required, use the following verb:

- DELETE_TP

If the invokable TP requires conversation-level security, you need to define user IDs and passwords that remote TPs can use to access SNAplus2 TPs. To define user IDs and passwords, check the current definitions, or delete user IDs and passwords when they are no longer required, use the following verbs:

- DEFINE_USERID_PASSWORD, QUERY_USERID_PASSWORD, DELETE_USERID_PASSWORD

To restrict the use of the TP to a specific list of authorized user IDs, check the current list of authorized user IDs, or delete a list of user IDs when it is no longer required, use the following verbs:

- DEFINE_SECURITY_ACCESS_LIST, QUERY_SECURITY_ACCESS_LIST, DELETE_SECURITY_ACCESS_LIST

CPI-C Communications

CPI-C applications use the same resources as APPC applications; the information in “APPC Communications”, applies to CPI-C as well as to APPC.

In addition, you can set up side information entries for use by CPI-C applications; each entry defines a particular partner application and the information required to access it. The local CPI-C application can then identify its partner application simply by the name of a side information entry, instead of having to specify explicit partner LU and TP names, mode name, and conversation security requirements. To define side information entries, check the current definitions, or delete entries when they are no longer required, use the following verbs:

- DEFINE_CPIC_SIDE_INFO, QUERY_CPIC_SIDE_INFO, DELETE_CPIC_SIDE_INFO

Managing PU Concentration

If the node supports PU concentration (the *pu_conc_supported* parameter on the DEFINE_NODE verb), to enable type 0-3 LUs on downstream computers to communicate with host systems using LUs defined on the SNAplus2 node, you must first define the following:

- A DLC, port, and LS from SNAplus2 to the downstream computer. For information about defining these components, see “Getting

Started". The LS must be defined with the following parameters:

```
solicit_sscp_sessions   =  NO
dspu_services    =  PU_CONCENTRATION
dspu_name =  the name of the PU serving the LUs on the downstream computer
pu_name  =  all zeros
```

- One or more type 0-3 LUs on the SNAplus2 node (and optionally an LU pool containing these LUs) for communications with the host. For information about defining LUs and LU pools, see "3270 Communications".

You then define the LUs on the downstream computer and map these to the LUs on the SNAplus2 node. To define the downstream LUs, use the following verbs:

- DEFINE_DOWNSTREAM_LU, DEFINE_DOWNSTREAM_LU_RANGE

To obtain information about the configuration or current status of downstream LUs or about the downstream PU that serves them, use the following verbs:

- QUERY_DOWNSTREAM_LU, QUERY_DOWNSTREAM_PU

To delete downstream LUs when they are no longer required, use the following verbs:

- DELETE_DOWNSTREAM_LU, DELETE_DOWNSTREAM_LU_RANGE

Managing DLUR

If the node supports DLUR (the *dlur_supported* parameter on the DEFINE_NODE verb), and LUs on the SNAplus2 node will be using DLUR to communicate with host systems, you need to define the PU on the local SNAplus2 node that owns these LUs. This is not the same as defining a PU for LUs that communicate directly with the host (which is done using the DEFINE_LS verb).

To define the PU, use the following verb:

- DEFINE_INTERNAL_PU

To obtain information about the PU, use the following verb:

- QUERY_PU

To define and manage the LUs associated with this PU, see “3270 Communications”, “RJE Communications”, or “LUA Communications”, earlier in this section.

To start the PU (to request an ACTPU from the host) in order to use the LUs or to stop it when applications are no longer using the LUs, use the following verbs:

- START_INTERNAL_PU, STOP_INTERNAL_PU

To delete the PU when it is no longer required, use the following verb:

- DELETE_INTERNAL_PU

To set up defaults to simplify DLUR configuration and reduce the information required on other DLUR verbs, use the following verb:

- DEFINE_DLUR_DEFAULTS

To obtain information about PUs and LUs currently using DLUR , or about the DLUS nodes they are using, use the following verbs:

- QUERY_DLUR_PU, QUERY_DLUR_LU, QUERY_DLUS

Managing TN Server

If TN3270 users will be using the TN server feature on a SNAplus2 node to communicate with host systems, you need to define the communications link to the host. For more information, see “Getting Started”. The definition of the LS to the host must include the name of a local PU to own the 3270 LUs and must have the *solicit_sscp_sessions* parameter set to AP_YES.

You then need to define LUs that can be used for 3270 emulation and optionally group these LUs into LU pools. For more information about defining LUs and users, see “3270 Communications”.

To define the TN3270 users that can access TN server and assign them to SNAplus2 3270 LUs, use the following verb:

- DEFINE_TN3270_ACCESS

To obtain information about the configuration of TN3270 users, use the following verb:

- QUERY_TN3270_ACCESS_DEF

To delete TN3270 users so that they can no longer use TN server for 3270

emulation, use the following verb:

- DELETE_TN3270_ACCESS

Managing SNA Management Services Functions

If applications running on SNAplus2 will be using the MS API to communicate with remote MS applications, you do not need to define any resources for this explicitly, because the node will locate the appropriate remote applications as required. However, you can define the resources explicitly if you want to specify a particular remote application to use.

To specify a default PU for use by NMVT-level applications (so that they access the NetView program at a specific host), use the following verb:

- DEFINE_DEFAULT_PU

To specify a focal point application for use by MDS-level applications (instead of enabling the remote focal point application to determine which nodes it manages), use the following verb:

- DEFINE_FOCAL_POINT

To obtain information about the focal point currently in use, or to delete a previously defined focal point, use the following verbs:

- QUERY_FOCAL_POINT, DELETE_FOCAL_POINT

To obtain information about active applications (NMVT-level or MDS-level) using MS functions, use the following verbs:

- QUERY_NMVT_APPLICATION, QUERY_MDS_APPLICATION

To obtain information about outstanding requests from MDS-level applications, or to obtain statistical information about previous requests, use the following verbs:

- QUERY_ACTIVE_TRANSACTION, QUERY_MDS_STATISTICS

Managing Access to the SNAplus2 System from the Host NetView Program

If you want to enable operators at the host NetView console to issue commands on the SNAplus2 computer using either the Service Point Command Facility (SPCF) or the HP-UX Command Facility (UCF), you need to define the access permissions for these operators.

To define these permissions and enable NetView operators to access SPCF or UCF or both, use the following verb:

- DEFINE_RCF_ACCESS

To check the permissions currently defined, use the following verb:

- QUERY_RCF_ACCESS

To prevent operators from using either SPCF or UCF, use the following verb:

- DELETE_RCF_ACCESS

To remove access to one function but leave the other available, use the following verb:

- DEFINE_RCF_ACCESS

Managing Diagnostics Settings

The SNAplus2 default setting for log messages is to log problem and exception messages but not audit messages, and to use central logging (messages from all servers sent to a central log file on the master server). Succinct logging is used (that is, logging of header parameters and message text, but not full details of cause and action for each message). The error log file, used for problem and exception messages, is `/var/opt/sna/sna.err`; the audit log file, used for audit messages if these are enabled, is `/var/opt/sna/sna.aud`. Each of these files is backed up and reset when the file sopt/snaize reaches 1 megabyte. The default settings for succinct logging, exception and audit logging, file names, and file sizes can all be overridden using NOF verbs, as described in the following information:

The verbs to manage central logging and global logging options apply to Windows clients as well as to HP-UX computers. However, other diagnostics settings on Windows clients are controlled by options in the Windows client initialization file, `sna.ini` and not by NOF verbs. For more information, refer to the *HP-UX SNAplus2 Administration Guide*.

To specify whether central logging is enabled, use the following verb:

- SET_CENTRAL_LOGGING

To specify whether exception messages or audit messages or both are logged, or to specify whether succinct logging or full logging is to be used, either to establish global default settings for all servers or to override the

defaults for a particular server, use the following verbs:

- SET_GLOBAL_LOG_TYPE, SET_LOG_TYPE

To change the file names or directories used for log messages or to change the size at which files are backed up and reset, use the following verb:

- SET_LOG_FILE

To check which server is currently defined as the central logger or to check whether central logging is enabled, use the following verbs:

- QUERY_CENTRAL_LOGGER, QUERY_CENTRAL_LOGGING

To check the types of messages being recorded or to check whether succinct logging or full logging is being used, either globally or on a particular server, use the following verbs:

- QUERY_GLOBAL_LOG_TYPE, QUERY_LOG_TYPE

To check the file, file size, or directory being used for a particular log type, use the following verb:

- QUERY_LOG_FILE

If you want to activate tracing to diagnose problems with connectivity components on a particular SNAplus2 node or to deactivate it after collecting the required data, use the following verbs:

- ADD_DLC_TRACE, REMOVE_DLC_TRACE

If you want to activate tracing to diagnose problems with other SNAplus2 kernel components or to deactivate it after collecting the required data, use the following verb:

- SET_TRACE_TYPE

If you want to activate tracing to diagnose problems with communications between clients and servers across the SNAplus2 LAN or to deactivate it after collecting the required data, use the following verb:

- SET_CS_TRACE

If you want to activate tracing to diagnose problems with communications between current-level and back-level computers (in a client-server system where you are in the process of migrating to a new level of the SNAplus2 software) or to deactivate it after collecting the

required data, use the following verb:

- SET_BCK_CS_TRACE

If you want to activate tracing to diagnose problems with the SNAplus2 TN server feature or to deactivate it after collecting the required data, use the following verb:

- SET_TN_SERVER_TRACE

The default files used for trace data are as follows:

- `/var/opt/sna/sna1.trc` and `/var/opt/sna/sna2.trc` for tracing on a particular computer
- `/var/opt/sna/snacs1.trc` and `/var/opt/sna/snacs2.trc` for LAN tracing
- `/var/opt/sna/snabcs1.trc` and `/var/opt/sna/snabcs2.trc` for back-level client-server tracing
- `/var/opt/sna/snatnsv1.trc` and `/var/opt/sna/snatnsv2.trc` for TN server tracing

If you want to use different files or directories for either of these trace types or to send all tracing of a particular type to one file instead of two files, use the following verb:

- SET_TRACE_FILE

To check the current settings for a particular trace type or to check the files used for a particular trace type, use the following verbs:

- QUERY_DLC_TRACE, QUERY_TRACE_TYPE, QUERY_CS_TRACE, QUERY_BCK_CS_TRACE, QUERY_TN_SERVER_TRACE, QUERY_TRACE_FILE

The verbs relating to client-server tracing can be issued on HP-UX client computers as well as on a server. For more information,see “SET_CS_TRACE”, “QUERY_CS_TRACE”, “SET_TRACE_FILE”, and “QUERY_TRACE_FILE”.

Managing Directory Entries

If the local node is a LEN node, you need to set up entries in the local node's directory to identify the adjacent nodes that SNAplus2 will communicate with and the LUs associated with these nodes. If a particular node contains a range of LUs with similar names, you can set

up wildcard entries in the directory to indicate that all LUs in the range are on the specified node.

To define a node and the LUs associated with it, use the following verb:

- DEFINE_ADJACENT_LEN_NODE

To obtain information about a specific node or LU entry in the database (however, this verb cannot be used to return information about wildcard entries), use the following verb:

- QUERY_DIRECTORY_ENTRY

To obtain information about a specific LU entry or wildcard entry in the database, use the following verb:

- QUERY_DIRECTORY_LU

To obtain statistical information about directory entries, use the following verb:

- QUERY_DIRECTORY_STATS

To delete a node and the LUs associated with it or to delete LUs from a node entry, use the following verb:

- DELETE_ADJACENT_LEN_NODE

If the local node is an end node communicating with a LEN node, you need to set up directory entries for the LEN node and its LUs, using the verbs described above. For communications with other node types, you do not need to set up directory entries because the node will locate these resources dynamically as required (and add them to the directory so that they can be used again).

However, you may want to set up entries for particular nodes or LUs so that the local node can communicate with these resources without having to search for them. Because setting up entries for particular nodes or LUs overrides the normal APPN resource location process, you can have problems at this node or at other nodes in the network if the definitions are not correct. Do not define explicit entries for resources at other nodes unless you are sure that the definitions are correct.

To define an individual node, LU, or wildcard entry for a range of LUs, use the following verb:

- DEFINE_DIRECTORY_ENTRY

To delete an individual node, LU, or wildcard entry from the directory,

use the following verb:

- DELETE_DIRECTORY_ENTRY

Delete only directory entries that were explicitly defined using the verbs described previously (these entries return an entry type of HOME on the QUERY_DIRECTORY_ENTRY verb). Do not use this verb to delete cached entries (entries that have been set up dynamically as a result of network searches).

Querying the Network Topology

To obtain information about the TGs to adjacent network nodes, use the following verb:

- QUERY_LOCAL_TOPOLOGY

Checking the Communications Path to a Remote LU

To check that a particular target LU can be accessed (that the node owning the LU is active and that there is a communications path to the LU), use the following verb:

- APING

Managing Servers on the SNAplus2 LAN

To obtain a list of servers (nodes) on the SNAplus2 LAN, use the following verb:

- QUERY_NODE_ALL

To obtain detailed information about a particular node, use the following verb:

- QUERY_NODE

To find out which servers are acting as the master configuration file server and backup master servers, use the following verb:

- QUERY_SNA_NET

To add new backup master servers to the list or to remove existing servers from the list so that they can no longer act as master servers, use the following verbs:

- ADD_BACKUP, DELETE_BACKUP

Managing Configuration File Header Information

To add a descriptive comment string to the domain configuration file, use the following verb:

- DEFINE_DOMAIN_CONFIG_FILE

To obtain information about the SNAplus2 version number for which the domain configuration file was created or about the comment string stored in it, use the following verb:

- QUERY_DOMAIN_CONFIG_FILE

There are no corresponding verbs for the node configuration file because the header information in the node configuration file is for SNAplus2 internal use only; do not attempt to modify it.

Managing HP-UX Resource Usage

To set a limit on the amount of kernel memory that SNAplus2 can use for internal data structures or to specify the maximum amount of memory available for STREAMS buffers, use the following verbs:

- SET_KERNEL_MEMORY_LIMIT, SET_BUFFER_AVAILABILITY

To obtain information about the current limits and usage, use the following verbs:

- QUERY_KERNEL_MEMORY_LIMIT, QUERY_BUFFER_AVAIALBILITY

NOF Indications

A NOF application can use the REGISTER_INDICATION_SINK verb to request information about changes to the SNAplus2 configuration or to the status of its resources. SNAplus2 then sends an indication message to the application each time a change occurs.

For a complete list of indications that an application can request, see Chapter 6, “NOF Indications.”

Except for CONFIG_INDICATION, NOF_STATUS_INDICATION, and SNA_NET_INDICATION, each indication is returned when a resource of the specified type changes its status. For example, if the application registers for DLC indications, SNAplus2 sends a DLC_INDICATION message to the application each time a DLC becomes active or inactive.

An indication returns summary information about the change that has occurred. If necessary, the application can then issue the appropriate QUERY_* verb to obtain more detailed information.

If the local node is short of resources, it can temporarily suppress indications and not send them to applications. When the resource shortage condition clears, and the local node subsequently generates an indication of a type that it has previously suppressed, it then sets a parameter on the indication to inform the application that one or more previous indications of this type have been lost. The application should then issue QUERY_* verbs for the appropriate resource type to determine the current state of resources.

For more information about registering to receive indications, see “REGISTER_INDICATION_SINK”. For more information about individual indications, see Chapter 6, “NOF Indications.”

Configuration Indications

An application can register to receive information about changes to the configuration of a particular target (the domain configuration file, a running node, or an inactive node). This enables it to keep track of changes that can be made by other NOF applications or by the administration programs. To do this, the application registers as for other indications, specifying CONFIG_INDICATION as the requested indication type.

No specific VCB structure is associated with this indication type. Instead, when a change to the configuration occurs, SNAplus2 indicates this change to the application by sending a copy of the completed VCB from the NOF verb that made the change.

For more information about configuration indications, see "CONFIG_INDICATION".

SNA Network File Indications

An application can register to receive information about changes to the SNA network file `sna.net` on the master server. This enables the application to keep track of changes to this file that can be made by other NOF applications or by the command-line administration program. To do this, the application registers as for other indications, specifying SNA_NET_INDICATION as the requested indication type.

Two VCB structures are associated with this indication type:

- ADD_BACKUP (indicating that a backup server has been added to the end of the file)
- DELETE_BACKUP (indicating that an unused backup server has been removed from the file)

Registering with a type of SNA_NET_INDICATION will return an ADD_BACKUP indication when a backup server is added or a DELETE_BACKUP indication when a server is deleted; the application does not need to register separately for each of these indications. In each case, the format of the indication is a copy of the completed VCB from the NOF verb that made the change.

For more information about SNA network file indications, see "SNA_NET_INDICATION".

NOF Status Indications

SNAplus2 sends a NOF status indication to a registered NOF application when the application can no longer access its target node or file (because the SNAplus2 software on the target computer has been stopped or because the communications path to this computer has been lost). If the application is registered to receive indications from the master configuration file, this indication is also returned if another server takes over as master (and therefore the target file is no longer the master configuration file).

The application does not need to register explicitly to receive this indication; SNAplus2 returns it to any application that has registered for any type of NOF indications on the appropriate target. The indication is returned on the callback routine that the application supplied to the REGISTER_INDICATION_SINK verb (or to the first REGISTER_INDICATION_SINK verb, if the application has issued more than one).

After the application receives an indication that the target has failed, all subsequent verbs using the relevant target handle will be rejected, except for DISCONNECT_NODE or CLOSE_FILE (to free the target handle). In addition, any registrations for indications on this target handle will be lost; in order to continue receiving indications when the target becomes available, the application must connect again to the target and register again for the required indications.

For more information about NOF status indications, see Chapter 6, “NOF Indications.”

2 Writing NOF Applications

Overview

This chapter describes how a NOF application:

- Uses the NOF API entry points
- Schedules asynchronous events
- Is compiled and linked to use the NOF API

This chapter also describes the following information:

- Target (node or file) for NOF verbs, and how they interact with the target
- Ordering and dependencies between NOF verbs
- NOF restrictions based on node configuration
- How to request single or multiple data entries with QUERY_* verbs

Description of the NOF API Entry Points

An application accesses the NOF API using the following entry point function calls:

`nof`

Issues a NOF verb synchronously. SNAplus2 does not return control to the application until verb processing has finished. All NOF verbs except REGISTER_INDICATION_SINK and UNREGISTER_INDICATION_SINK can be issued through this entry point.

An application can use only this entry point if the application can suspend while waiting for SNAplus2 to completely process a verb.

The `nof` entry point is defined in the NOF header file `nof_c.h`.

`nof_async`

Issues a NOF verb asynchronously. SNAplus2 returns control to the application immediately, with a returned value indicating whether verb processing is still in progress or has completed. If the returned value indicates that verb processing is still in progress, SNAplus2 uses an application-supplied callback routine to return the results of the verb processing. In cases when SNAplus2 is able to completely process the request, the callback routine will not be invoked.

All NOF verbs can be issued through this entry point. The REGISTER_INDICATION_SINK and UNREGISTER_INDICATION_SINK verbs must be issued through this entry point.

An application must use this entry point if either of the following conditions is true:

- The application needs to receive NOF indications.
- The application cannot suspend while waiting for SNAplus2 to completely process a verb.

The nof_async entry point is defined in the NOF header file nof_c.h.

nof_async callback routine

When using the asynchronous NOF API entry point, the application must supply a pointer to a callback routine. SNAplus2 uses this callback routine both for completion of a verb and also for returning NOF data and status indications.

Synchronous Entry Point: nof

An application uses the nof entry point to issue a NOF verb synchronously. SNAplus2 does not return control to the application until verb processing has finished.

Function Call

```
void nof (
        AP_UINT32       target_handle,
        void *          nofvcb
        );
```

Supplied Parameters

An application supplies the following parameters when it uses the nof entry point:

target_handle An identifier that the application uses to identify the target SNAplus2 node or file. This parameter is supplied in one of the following ways:

- For the following verbs, this parameter is not supplied; set it to 0 (zero). If the verb completes successfully, SNAplus2 returns the target handle as one of the VCB parameters. The application then uses the target handle for subsequent verbs.
  - — CONNECT_NODE (to access a running node, or to access the node on a server where the SNAplus2 software is started but the node is not yet started)
  - — OPEN_FILE (to access the domain configuration fileor the SNA network data file)

- For the following verbs, the application supplies a null value:
  - — QUERY_NODE_ALL (to obtain a list of running nodes)
  - — QUERY_CENTRAL_LOGGER
- For all other NOF verbs, the application supplies the value that was returned on the CONNECT_NODE or OPEN_FILE verb.

nofvcb Pointer to a Verb Control Block (VCB) that contains the parameters for the verb being issued. The VCB structure for each verb is described in Chapters 3, 4, and 5. These structures are defined in the NOF API header file nof_c.h.

Returned Values

The nof entry point does not have a return value. When the call returns, the application should examine the return code in the VCB to determine whether the verb completed successfully and to determine parameters it needs for further verbs. In particular, when the CONNECT_NODE or OPEN_FILE verb completes successfully, the VCB contains the *target_handle* that the application should use when issuing subsequent verbs.

Using the Synchronous Entry Point

Only one synchronous verb can be outstanding at any time for each target handle. A synchronous verb fails with the primary return code AP_STATE_CHECK and secondary return code AP_SYNC_PENDING if another synchronous verb for the same target handle is in progress.

Asynchronous Entry Point: nof_async

An application uses nof_async to issue a NOF verb asynchronously. The application also supplies a pointer to a callback routine. SNAplus2 returns control to the application immediately with a returned value that indicates whether verb processing is still in progress or has completed. In most cases, verb processing is still in progress when control returns to the application. In these cases, SNAplus2 uses the application-supplied callback routine to return the results of the verb processing at a later time. In some cases, verb processing is complete

when SNAplus2 returns control to the application, so SNAplus2 does not use the application's callback routine.

Function Call

```
AP_UINT16      nof_async(
                         AP_UINT32      target_handle,
                         void *         nofvcb,
                         NOF_CALLBACK   (*comp_proc),
                         AP_CORR        corr
                        );

typedef void (*NOF_CALLBACK) (
                              AP_UINT32      target_handle,
                              void *         nofvcb,
                              AP_CORR        corr
                              AP_UINT32      indic_length
                              );

typedef union ap_corr  {
                         void *         corr_p;
                         AP_UINT32      corr_l;
                         int            corr_i;
                       } AP_CORR;
```

For more information about the parameters in the NOF_CALLBACK structure, see “The Callback Routine Specified on the nof_async Entry Point”.

Supplied Parameters

An application supplies the following parameters when it uses the nof_async entry point:

target_handle This parameter is supplied in one of the following ways:

- For the following verbs, this parameter is not used; set it to 0 (zero). If the verb completes successfully, SNAplus2 returns the target handle as one of the VCB parameters. The application then uses the target handle for subsequent verbs.

  — CONNECT_NODE (to access a running node, or to access the node on a server where the SNAplus2 software is started but the node is not

yet started)

— OPEN_FILE (to access the domain configuration fileor the SNA network data file)

- For the following verbs, the application supplies a null value:

  — QUERY_NODE_ALL (to obtain a list of running nodes)

  — QUERY_CENTRAL_LOGGER

- For all other NOF verbs, the application supplies the value that was returned on the CONNECT_NODE or OPEN_FILE verb.

nofvcb Pointer to a Verb Control Block (VCB) that contains the parameters for the verb being issued. The VCB structure for each verb is described in Chapters 3, 4, and 5. These structures are defined in the NOF API header file `nof_c.h`.

comp_proc The callback routine that SNAplus2 will call when the verb completes. For more information about the requirements for a callback routine, see "The Callback Routine Specified on the nof_async Entry Point".

corr An optional correlator for use by the application. This parameter is defined as a C union so that the application can specify any of three different parameter types: pointer, unsigned long, or integer.

SNAplus2 does not use this value, but passes it as a parameter to the callback routine when the verb completes. This value enables the application to correlate the returned information with its other processing.

Returned Values

The asynchronous entry point returns one of the following values:

`AP_COMPLETED`

The verb has already completed. The application can examine the parameters in the VCB to determine whether the verb completed successfully. SNAplus2

does not call the supplied callback routine for this verb.

AP_IN_PROGRESS

The verb has not yet completed. The application can continue with other processing, including issuing other NOF verbs, provided that they do not depend on the completion of the current verb. However, the application should not attempt to examine or modify the parameters in the VCB supplied to this verb.

SNAplus2 calls the supplied callback routine to indicate when the verb processing completes. The application can then examine the VCB parameters.

Using the Asynchronous Entry Point

When using the asynchronous entry point, note the following:

- If an application specifies a null pointer in the *comp_proc* parameter, the verb will complete synchronously (as though the application issued the verb using the synchronous entry point).
- If the call to nof_async is made from within an application callback, specifying a null pointer in the *comp_proc* parameter is not permitted. In such cases, SNAplus2 rejects the verb with a primary return code value of AP_PARAMETER_CHECK and a secondary return code value of AP_SYNC_NOT_ALLOWED.
- The application must not attempt to use or modify any parameters in the VCB until the callback routine has been called.
- Multiple verbs do not necessarily complete in the order in which they were issued. In particular, if an application issues an asynchronous verb followed by a synchronous verb, the completion of the synchronous verb does not guarantee that the asynchronous verb has already completed.

The Callback Routine Specified on the nof_async Entry Point

When using the asynchronous NOF API entry point, the application must supply a pointer to a callback routine. SNAplus2 uses this callback routine both for completion of a verb and also for returning NOF indications. (The REGISTER_INDICATION_SINK verb is also issued as

an asynchronous verb that specifies a callback routine; the callback is called each time the indication is received. For other NOF verbs, an indication is received when the verb completes.) The application must examine the *opcode* parameter in the VCB to determine which event is contained in the callback routine.

This section describes how SNAplus2 uses the callback routine and the functions that the callback routine must perform.

Callback Function

```
NOF_CALLBACK (*comp_proc);
typedef void (*NOF_CALLBACK) (
                              AP_UINT32      target_handle,
                              void *         nofvcb,
                              AP_CORR        corr
                              AP_UINT32      indic_length
                             );
typedef union ap_corr  {
                         void *         corr_p;
                         AP_UINT32      corr_l;
                         int            corr_i;
                       } AP_CORR;
```

Supplied Parameters

SNAplus2 calls the callback routine with the following parameters:

target_handle For NOF indications, SNAplus2 passes the target handle that was supplied with the REGISTER_INDICATION_SINK verb. For completion of verbs, this parameter is undefined.

nofvcb One of the following:

- For NOF indications, a pointer to a VCB supplied by SNAplus2.
- For completion of verbs, a pointer to the VCB supplied by the application. The VCB now includes the returned parameters set by SNAplus2.

corr The correlator value supplied by the application. This value enables the application to correlate the returned information with its other processing.

The callback routine need not use all of these parameters (except as

described in “Using the Callback Routine for Indications”). The callback routine can perform all the necessary processing on the returned parameters, or it can simply set a variable to inform the NOF application that the verb has completed.

Returned Values

The callback function does not return any values.

Using the Callback Routine for Indications

Although the application allocates the VCBs for NOF verbs, SNAplus2 allocates the VCBs for indications. Therefore, the application has access to the VCB information only from within the callback routine; the VCB pointer that SNAplus2 supplies to the callback routine is not valid outside the callback routine. The application must either complete all the required processing from within the callback routine, or it must make a copy of any VCB data that it needs to use outside this routine.

In the event that the NOF application needs to make a copy of the data supplied on the callback routine and the NOF application is using a signal-based scheduling mode, an operating system limitation may prohibit memory allocation. In this case, some memory must be preallocated before the REGISTER_INDICATION_SINK verb is issued. For more information about signal-based scheduling mode, see “Signal-Based Scheduling Mode”.

Scope of Target Handle

Each application that needs to use NOF must issue the CONNECT_NODE verb to obtain its own handle. No two NOF applications can use the same NOF target handle.

In particular, if the application that issued CONNECT_NODE later forks to create a child process, the child process cannot issue any NOF verbs that use the target handle obtained by the parent process. However, the child process can issue another CONNECT_NODE to obtain its own target handle.

Scheduling Asynchronous Events

The method that an application uses to schedule asynchronous events depends on which of the following types of application it is:

Single-threaded applications

> Applications that are based around a single main thread of execution to receive and process requests

Multithreaded applications

> Applications that can have several threads of execution receiving and processing requests

Motif applications

> Applications that use the Motif interface and for which the application code consists mainly of callbacks from the Motif libraries

Single-Threaded Applications

To schedule asynchronous events, single-threaded applications can use either the application scheduling mode or the signal-based scheduling mode.

Application Scheduling Mode

Application scheduling mode gives the application full control over event scheduling from different sources by integrating SNA callbacks with the application's main processing loop. The application calls an SNA event handler entry point in the SNA library when work is available for the library to process, and the handler makes the necessary callbacks to the application.

NOTE SNA callbacks can also be made from within other calls into the SNA library.

To receive notification of SNA events, the application accesses the file descriptor on which SNA events arrive. The application then calls the SNA event handler to process events received on this file descriptor and

generate any subsequent callbacks. For more information about callbacks, see “The Callback Routine Specified on the nof_async Entry Point”.

The application scheduling mode assumes that the application is structured as a main loop consisting of a call to either `select` or `poll` followed by code to process event information returned by the `select` or `poll` call.

To use application scheduling mode, incorporate the following steps into your application code:

Step 1. Indicate that you want to use application scheduling mode by adding the following function call to your code before the first call into any SNA library:

```
SNA_USE_FD_SCHED();
```

The `SNA_USE_FD_SCHED` call has no error return values.

Step 2. Add code to obtain and track any changes to the SNA file descriptor. Call the following function from your code before any call to `select` or `poll` that can be used to detect SNA events:

```
Fd= SNA_GET_FD();
```

The `SNA_GET_FD` call returns either a valid file descriptor or a `-1` indicating that no valid SNA file descriptor is available.

Step 3. If the return is valid, the application then must register for “read” events on the SNA file descriptor in the `select` or `poll` call. When an event is detected, the application must call the SNA event handler.

Step 4. To call the SNA event handler, add the following function call to your code:

```
SNA_EVENT_FD();
```

The `SNA_EVENT_FD` call has no error return values.

Signal-Based Scheduling Mode

Signal-based scheduling mode provides binary compatibility for existing SNAplus2 Release 4 and SNAplus2 Release 5; applications. It is not recommended for new applications because support for signal-based scheduling mode may be discontinued in future versions of SNAplus2.

Asynchronous verbs are implemented by making callbacks to the

application from signal catcher context. The disadvantages of signal-based scheduling mode are:

- Applications that receive work from multiple sources are difficult to write.
- Applications are required to use HP-UX V.3 signal calls.
- Applications can use only a subset of system calls made from signal catcher context and therefore from the API callback context.
- Signaling mode is not thread-safe

Use of Signals To detect work arriving from other components of SNAplus2, the NOF library makes use of the `SIGPOLL` signal. Applications can require the `SIGPOLL` signal for other purposes. Therefore, the library daisy-chains to any existing signal catchers. The first verb issued by the application initializes the library's signal catcher.

When the `SIGPOLL` signal arrives, either it indicates work for SNAplus2 or it was generated for some other reason. The NOF library's signal catcher processes the signal and then calls the previous signal catcher to allow the signal to be used for other purposes.

The following restrictions apply to `SIGPOLL` usage:

- Applications must not permanently disable this signal. The `sighold` and `sigrelse` signal calls can be used to protect a critical region of code, provided that no SNA verbs are issued between `sighold` and `sigrelse`.
- Applications must preserve the address of the SNAplus2 signal catchers returned by the `sigset` call when adding signal catchers to applications. Applications must then arrange to call these routines from within the new signal catchers.
- Applications must not use the POSIX signal mechanism `SIGACTN`, which can cause problems in SNAplus2 applications.

HP-UX Sleep Call The HP-UX operating system call `sleep` can cause problems if applications or their libraries also use signal catchers. Because the SNAplus2 NOF library operates using a signal catcher, do not use the `sleep` call within a NOF application that uses signal-based scheduling. If necessary, you can use the `alarm` with a timer to provide the same function.

Multithreaded Applications

SNAplus2 API libraries are available for linking with multithreaded applications. When you develop applications to operate in a multithreaded environment, the following restrictions apply:

- When an application uses the asynchronous entry point, the application is required to maintain the consistency of its data structures when callbacks are invoked. Consistency of data structures can be maintained using the multithreading `lock` or `mutex` facilities. The callbacks are made in the context of a separate thread created and managed from within the SNAplus2 API library. Since asynchronous callbacks run using a separate thread, the application is not required to provide a source of scheduling to enable the callbacks. Do not use application scheduling mode in a multithreaded application.
- The application must perform any required cleanup processing (for example, issuing UNREGISTER_INDICATION_SINK and issuing DISCONNECT_NODE or CLOSE_FILE) before a thread terminates. The NOF library does not maintain any correlation between threads and NOF verb usage and does not perform this processing automatically when a thread terminates.

For the HP-UX 10.20 operating system, kernel threads are not supported.

Motif Applications

Applications that use the Motif interface and whose code consists mainly of callbacks from the Motif libraries are required to add SNA events to the main Xt library scheduling loop. The SNA events allow the SNAplus2 library to run callbacks in order to process asynchronous verb completions.

Add the following lines to your code before the first call into any SNA library:

```
#include <Xt.h>
int app_context;
...
XtAppInitialize(app_context...)
...
SNA_USE_XT_SCHED(app_context);
```

The `SNA_USE_XT_SCHED` call has no return values. It calls the

`XtAppAddInput` function to register the SNA work sources. When work subsequently arrives on the SNA file descriptor, the Xt library scheduling loop calls the SNA event handler, which then makes any required API callbacks to the application.

NOTE SNA callbacks can also be made from within other calls into the SNA library

Compiling and Linking the NOF Application

Applications are compiled with different options in order to select one of the scheduling modes described in “Scheduling Asynchronous Events”.

NOTE

Applications that use asynchronous API callbacks must either be built as multithreaded applications or include support for the application scheduled mode. Motif applications must include the code fragment described in “Motif Applications”.

Applications linked with previous versions of SNAplus2 that used signal-based scheduling mode and the single-threaded library, `libmgr.sl`, will continue to work in the current version of SNAplus2. However, this library is provided only for back-compatibility. New applications should not use this library.

You can use the following options when you link your program (for specific examples of link commands, see “Linking Motif Applications and Applications That Use Application Scheduled Mode” and “Linking Multithreaded Applications”):

`-I`	Indicates the include path.
`-L`	Indicates the directory of the library or libraries to be used when linking the application.
`-l`	Each `-l` indicates the name of a library.

Linking Motif Applications and Applications That Use Application Scheduled Mode

For HP-UX operating systems, link with the following options:

```
cc -I /opt/sna/include -L /opt/sna/lib -lnof -lsna
```

Linking Multithreaded Applications

For HP-UX operating systems, linking options depend on the version of the operating system:

- For HP-UX 10.20 (only DCE threads are supported), link with the

following options:

```
cc -I /opt/sna/include -L /opt/sna/lib -lnof -lmgrdce -ldce
```

- For HP-UX 11.0 using DCE threads, link with the following options:

```
cc -I /opt/sna/include -L /opt/sna/lib -lnof -lsna -ldce
```

- For HP-UX 11.0 using Kernel threads, link with the following options:

```
cc -I /opt/sna/include -L /opt/sna/lib -lnof -lsna -lpthread
```

Target For NOF Verbs

A NOF verb can be directed to any of the following targets:

- A running node (to manage the node's resources)
- The node on a server where the SNAplus2 software is running but where the node has not been started (to start the node, to query the node's stored configuration, or to modify the configuration so that the changes take effect when the node is restarted)
- The domain configuration file (to manage domain resources)
- The `sna.net` file (to manage the SNAplus2 servers that can act as backup masters if the master server is not available)

The target for a particular NOF verb is identified by the *`target_handle`* parameter used on the NOF call. An application acquires a target handle using different NOF verbs depending on the target, as follows:

Running node or node on running server

> The application issues CONNECT_NODE, specifying the name of the required node, with a null target handle; SNAplus2 returns a target handle for this node as one of the VCB parameters for CONNECT_NODE.

Domain configuration file

> The application issues OPEN_FILE with a null target handle; SNAplus2 returns a target handle for the file as one of the VCB parameters for OPEN_FILE.

`sna.net` file

> The application issues OPEN_FILE with a null target handle; SNAplus2 returns a target handle for the file as one of the VCB parameters for OPEN_FILE.

Some NOF verbs can be issued only to particular target types:

- DEFINE_NODE cannot be issued to a running node; it must be issued to a server where the node is not running.
- Verbs associated with node resources, such as DEFINE_LOCAL_LU, must be issued to a node.

- START_* and STOP_* verbs, to start and stop node resources, must be issued to a running node.
- Verbs associated with domain resources, such as DEFINE_EMULATOR_USER, must be issued to the domain configuration file.
- Different QUERY_* verbs return information about the definition of a resource, on its current status, or on both definition and status. Status information can only be obtained from a running node. Verbs that return only status information cannot be issued to an inactive node, and verbs that return both definition and status will return only definition information when issued to an inactive node. For example, QUERY_PARTNER_LU_DEFINITION can be issued either to an inactive node (to determine the stored configuration) or to a running node (to determine the current definition). However, QUERY_PARTNER_LU (which returns information about the LU's current sessions) can be issued only to a running node. QUERY_LS (which returns both the definition of the LS and its current status) can be issued either to an inactive node or to a running node, but status information is not returned if you issue it to an inactive node. The description of each QUERY_* verb in Chapter 3, “NOF API Verbs (ACTIVATE_SESSION to OPEN_FILE),” includes information about the valid target types for the verb.
- Verbs associated with managing backup master servers (ADD_BACKUP, DELETE_BACKUP, QUERY_SNA_NET, REGISTER_INDICATION_SINK and UNREGISTER_INDICATION_SINK for SNA network file indications) must be issued to the `sna.net` file.

Processing Modes

Each target handle used by an application has an associated processing mode that can be modified with the NOF verb SET_PROCESSING_MODE. The mode controls file locking and access permissions for the application. The following modes are available:

`AP_READ_ONLY` Only QUERY_* verbs are enabled in this mode. All other verbs, which modify the configuration or status of a resource, will be rejected.

This is the default mode when the target handle is first assigned; it enables the application to check the configuration or status of a resource but not to change

it.

`AP_READ_WRITE` All NOF verbs are enabled in this mode, including those that change a resource's configuration or status.

`AP_COMMIT` This mode is only available if the target handle identifies the domain configuration file (not when issuing verbs to a node). It obtains a lock on the file so that only this application can access it; this file lock ensures that the file will not be modified by any other process during a sequence of verbs issued by this application. The file lock also ensures that no changes are made to the file until the complete sequence of verbs has been issued (until the application changes from `AP_COMMIT` mode to one of the other modes).

Because this mode prevents any other program from accessing the file, it should be used only for as long as necessary. The application should immediately issue all the verbs that it requires to modify the file and then change to one of the other modes.

If the file lock cannot be obtained (for example, because another program is currently modifying the file), the SET_PROCESSING_MODE verb will fail.

Ordering and Dependencies between NOF Verbs

The main restriction on the order of NOF verbs is that the first reference to a particular resource must be in a DEFINE_* verb for that resource. This leads to the following dependencies:

- When creating a new node configuration file, the first verb issued must be DEFINE_NODE.
- A DLC must be defined before any port that refers to it.
- A port must be defined before any LS or CN that refers to it.
- A COS must be defined before any mode that refers to it.
- A PU name must be defined (as part of an LS definition) before a dependent LU that refers to this PU.
- An LU must be defined before an LU pool that includes it.
- A downstream PU name (as part of an LS definition) and a host LU must be defined before a downstream LU that refers to them.
- A resource must be defined before a START_* verb refers to it, and must be started before a STOP_* verb refers to it.

In addition, when modifying a running node, using a DEFINE_* verb a second time (to modify the previous definition) is not always valid. For some of these verbs, a second definition is never valid (the resource must be deleted and then defined again); for others, a second definition is valid only if the resource is currently inactive. The descriptions of individual DEFINE_* verbs in Chapter 3, “NOF API Verbs (ACTIVATE_SESSION to OPEN_FILE),” provide information about whether a second definition is valid. When modifying the domain configuration file, a second DEFINE_* verb can always be used to modify a previous definition.

When creating a new node configuration file, the first verb issued must be DEFINE_NODE. This must be followed by DEFINE_* and SET_* verbs for all the resources associated with the node.

In the domain configuration file, there is no restriction on the ordering of domain resource records.

NOF Restrictions Based on Node Configuration

The DEFINE_NODE verb includes parameters that define the range of functions supported by a node. Several NOF verbs relate to optional functions that a node can or can not support; these verbs are valid only when issued to a node that supports the relevant functions.

This section summarizes the optional functions that affect which NOF verbs can be used. For more information about these functions, see "DEFINE_NODE".

APPN LEN Node Restrictions

The SNAplus2 local node can be an APPN end node or a LEN node.

The following NOF verbs are only valid at a network node or end node; the primary return code AP_FUNCTION_NOT_SUPPORTED is returned if you attempt to issue them at a LEN node.

- DEFINE_CN
- DELETE_CN
- QUERY_CN
- QUERY_CN_PORT

Multiple Domain Support (MDS) Restrictions

The local node can be run with or without Multiple Domain Support (MDS). The following NOF verbs are only valid at a node running with MDS; the primary return code AP_FUNCTION_NOT_SUPPORTEDis returned if you attempt to issue them at a node without MDS.

- QUERY_ACTIVE_TRANSACTION
- QUERY_MDS_APPLICATION
- QUERY_MDS_STATISTICS

PU Concentration and DLUR Restrictions

The local node can be run with or without support for PU concentration or DLUR or both.

The following NOF verbs are valid only if the node is running with PU concentration enabled; the primary return code AP_FUNCTION_NOT_SUPPORTED is returned if you attempt to issue them at a node without PU concentration.

- DEFINE_DOWNSTREAM_LU, DEFINE_DOWNSTREAM_LU_RANGE
- DELETE_DOWNSTREAM_LU, DELETE_DOWNSTREAM_LU_RANGE

The following NOF verbs are valid only if the node is running with PU concentration or DLUR or both enabled; the primary return code AP_FUNCTION_NOT_SUPPORTED is returned if you attempt to issue them at a node without either of these two functions.

- QUERY_DOWNSTREAM_LU, QUERY_DOWNSTREAM_PU

DLUR Restrictions

The following NOF verbs are valid only if the node is running with DLUR enabled; the primary return code AP_FUNCTION_NOT_SUPPORTED is returned if you attempt to issue them at a node without DLUR.

- DEFINE_DLUR_DEFAULTS
- DEFINE_INTERNAL_PU, DELETE_INTERNAL_PU
- START_INTERNAL_PU, STOP_INTERNAL_PU
- QUERY_DOWNSTREAM_LU, QUERY_DOWNSTREAM_PU
- QUERY_DLUR_LU, QUERY_DLUR_PU, QUERY_DLUS

List Options For QUERY_* Verbs

A NOF application can obtain information about a particular SNAplus2 resource by issuing a QUERY_* verb for the appropriate resource type. For example, it can obtain information about the configuration of an LS by issuing QUERY_LS. These verbs can either return information about a specific resource (for example, the configuration of a particular LS) or about many resources of the same type (for example, a summary of all configured LSs), depending on the options used. In addition, some QUERY_* verbs have the option of returning either summary or detailed information about the specified resources. This section explains how to use these options.

Obtaining Information about a Single Resource or Multiple Resources

You can think of the information returned by QUERY_* verbs as being stored in the form of a list, ordered according to the name of the resource. For example, the information returned by QUERY_LS is in order of LS name. The normal order of the list (for compatibility with SNMP list ordering) is as follows:

- By name length (shortest name first)
- By ASCII lexicographical ordering for names of the same length

Where the list ordering differs from this (for example, where the list is ordered by a numeric value), this difference is indicated in the individual verb descriptions in Chapter 4, “NOF API Verbs (QUERY Verbs).”

This means that an application can obtain information about multiple resources by requesting the complete list or a specified part of it. The following parameters on a QUERY_* verb determine which entries from the list are returned:

buf_size	Size of the data buffer that the application supplies to receive the returned information.
num_entries	Maximum number of resources for which information should be returned. The application can specify 1 to request a specific entry rather than a range, a number greater than 1 to request a range, or 0 (zero) to request as many entries as possible.

list_options The position in the list of the first entry required:

- First entry in the list
- Entries starting from a specific named entry
- Entries starting from the next entry after a specific named entry. (The name specified gives the starting position according to the list ordering and need not exist in the list; for example, if the list contains entries NODEA, NODEB, NODED, NODEF, and the application requests entries starting from the first entry after NODEC, the first entry returned is NODED.)

In addition, if the *list_options* parameter does not request starting from the first entry, the name of a specific entry in the list is used to indicate the starting position for the required entries.

The number of entries returned is the smallest of the following values:

- The *num_entries* parameter, if this is nonzero
- The maximum number of entries that the supplied data buffer can hold
- The number of entries between the specified starting position and the end of the list

In addition, the verb returns information about the total number of entries available and the size of the buffer that would be required to return all the entries at once. If the application has not yet received all the information it requires, it can then issue further verbs to obtain the remaining information.

These options enable the application to manage the information it receives, as follows:

- To obtain a specific entry, it sets the index value to the name of that entry, *list_options* to indicate “start from the named entry”, *buf_size* to at least the size of a single entry, and *num_entries* to 1.
- To obtain a complete list a few entries at a time, it first sets *list_options* to indicate “start from beginning of list”, and uses either *buf_size* or *num_entries* to limit the amount of information returned. If the returned values indicate that there is more information available, it then issues another verb with *list_options* indicating “start from the following entry” and sets the index value to the name of the last entry received; this second verb then returns the

next section of the list. The application repeats this process until it has received all the required entries.

Obtaining Summary or Detailed Information

Some QUERY_* verbs provide the option of returning either summary or detailed information about the specified resources. For example, QUERY_LOCAL_LU can return just the LU name and LU alias (summary information) or can also return additional information such as the LU address and session limit (detailed information). The description of each QUERY_* verb in Chapter 4, "NOF API Verbs (QUERY Verbs)," indicates whether the verb includes the option of returning summary or detailed information.

For the verbs that provide this option, the *list_options* parameter is used to indicate whether summary or detailed information is required, as well as the starting position within the list. To specify these options, you combine two values using a logical OR operation (one value to specify the starting position in the list and one value to specify whether summary or detailed information is required) and set the *list_options* parameter to the combination of these two values. For verbs that do not provide this option, you simply set *list_options* to a single value to indicate the starting position in the list.

3 NOF API Verbs (ACTIVATE_SESSION to OPEN_FILE)

Overview

The NOF API verbs are described in alphabetic order in this and the following two chapters.

These chapters provide the following information for each NOF API verb:

- Description of the verb's purpose and usage
- Whether the verb can be issued to an active node, an inactive node, the domain configuration file, or the SNA network data file (unless otherwise stated, verbs may be issued either to an active node or to an inactive node)
- Verb control block (VCB) structure, as defined in the NOF API header file `nof_c.h`
- Parameters supplied to the verb by the application
- Parameters returned to the application
- Error return codes for unsuccessful execution

The error return codes described are specific to each verb. Additional return codes, which are common to all NOF API verbs, are described in Appendix A, “Common Return Codes.”

NOF API indications, which the application can accept by registering using the REGISTER_INDICATION_SINK verb, are described separately in Chapter 6, “NOF Indications.”

ACTIVATE_SESSION

The ACTIVATE_SESSION verb requests SNAplus2 to activate a session between the local LU and a specified partner LU, using a specified mode. You must issue an INITIALIZE_SESSION_LIMIT verb before issuing an ACTIVATE_SESSION verb, unless *cnos_permitted* is set to AP_YES.

This verb must be issued to a running node.

VCB Structure

```
typedef struct activate_session
{
      AP_UINT16        opcode;          /* verb operation code             */
      unsigned char    reserv2;         /* reserved                        */
      unsigned char    format;          /* reserved                        */
      AP_UINT16        primary_rc;      /* primary return code             */
      AP_UINT32        secondary_rc;    /* secondary return code           */
      unsigned char    lu_name[8];      /* local LU name                   */
      unsigned char    lu_alias[8];     /* local LU alias                  */
      unsigned char    plu_alias[8];    /* partner LU alias                */
      unsigned char    mode_name[8];    /* mode name                       */
      unsigned char    fqplu_name[17];  /* fully qualified partner LU name */
      unsigned char    polarity;        /* requested session polarity      */
      unsigned char    session_id[8];   /* session ID                      */
      unsigned char    cnos_permitted;  /* is implicit CNOS permitted?     */
      unsigned char    reserv4[15];     /* reserved                        */
 ACTIVATE_SESSION;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_ACTIVATE_SESSION

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead

of its LU name, set this parameter to 8 binary zeros.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. This parameter is used only if *lu_name* is set to zeros.

If both the LU name and the LU alias are set to all zeros, the verb is forwarded to the LU associated with the CP (the default LU).

plu_alias

LU alias of the partner LU. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the partner LU is defined by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros.

mode_name

Name of the mode to be used by the LUs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. This parameter is used only if the *plu_alias* field is set to zeros; it is ignored if *plu_alias* is specified.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

polarity

The polarity for the session. Possible values are:

```
AP_POL_EITHER
AP_POL_FIRST_SPEAKER
```

AP_POL_BIDDER

If AP_POL_EITHER is set, ACTIVATE_SESSION activates a first speaker session if available, otherwise a bidder session is activated. If AP_POL_FIRST_SPEAKER or AP_POL_BIDDER is set, ACTIVATE_SESSION only succeeds if a session of the requested polarity is available.

cnos_permitted

Indicates that CNOS processing is permitted. Possible values are:

AP_YES

CNOS processing is permitted.

AP_NO

CNOS processing is not permitted.

If the activation of a new session is not possible because the session limits for the specified mode are reset, and this parameter is set to AP_YES, implicit CNOS processing will initialize the session limits. Execution of this command is suspended while CNOS processing is active.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Possible values are:

AP_AS_NEGOTIATED

The session was activated successfully; the session limit defined for the mode was negotiated during the activation process.

AP_AS_SPECIFIED

The session was activated successfully; the session limit was not changed.

session_id The 8-byte identifier of the activated session.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_EXCEEDS_MAX_ALLOWED

The session cannot be activated, because this would exceed the current session limit for this LU-LU-mode combination.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined local LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined local LU name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name, or the *plu_alias* parameter did not match any defined partner LU name.

AP_INVALID_CNOS_PERMITTED

The value specified in the *cnos_permitted* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Activation Failure

If the verb does not execute because of other errors, SNAplus2 returns one of the following parameters.

primary_rc Possible values are:

AP_ACTIVATION_FAIL_NO_RETRY

The session could not be activated because of a

condition that requires action (such as a configuration mismatch or a session protocol error). Check the SNAplus2 log file for information about the error condition, and correct it before retrying this verb.

`AP_ACTIVATION_FAIL_RETRY`

The session could not be activated because of a temporary condition (such as a link failure). Retry the verb, preferably after a timeout to allow the condition to clear. Check the SNAplus2 log file for information about the error condition.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

ADD_BACKUP

An application uses this verb to add a server to the list of backup master servers in the sna.net file, so that this server can act as the master configuration file server if the current master becomes inactive. The new server is added to the end of the list, so that it will only become the master if all the other servers listed in the file are inactive.

This verb must be issued to the sna.net file.

VCB Structure

```
typedef struct add_backup
{
      AP_UINT16        opcode;             /* verb operation code            */
      unsigned char    reserv2;            /* reserved                       */
      unsigned char    format;             /* reserved                       */
      AP_UINT16        primary_rc;         /* primary return code            */
      AP_UINT32        secondary_rc;       /* secondary return code          */
      unsigned char    backup_name[64];    /* name of backup server to add   */
      unsigned char    reserv4[4];         /* reserved                       */
} ADD_BACKUP;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_ADD_BACKUP
backup_name	The name of the server being added to the list of backup servers.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK
secondary_rc	Not used.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_DUPLICATE_RECORD

The server name specified is already listed in the file.

AP_INVALID_TARGET

The target handle on the NOF API call specified a configuration file or a node. This verb must be issued to the sna.net file.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

ADD_DLC_TRACE

This verb specifies tracing on SNA messages sent on a DLC. It can be used to activate tracing on a particular DLC, port, or LS, or on a particular session on a specified LS, and to specify which types of messages are to be traced. It can also be used to activate tracing on all DLCs, ports, and LSs. For more information about how to use SNAplus2 tracing, see the *HP-UX SNAplus2 Administration Guide.*

If multiple ADD_DLC_TRACE verbs relating to the same resource are issued, a message will be traced if it matches any of the verbs currently active. For example:

- If you issue a verb to trace all messages for a port and its LSs, and then issue a second verb to trace only messages with a specified LFSID for one of the LSs owned by the port, all messages for the LS will continue to be traced (because they match the first verb). If you then use REMOVE_DLC_TRACE to remove tracing for the port, messages on the LS with the specified LFSID will continue to be traced (because they match the second verb which is still active), but other messages on this LS will not be traced.
- If you issue a verb to trace XID messages on all resources, and then issue a second verb to trace SC and DFC messages on a particular LS, all three message types will be traced for this LS.

VCB Structure

```
typedef struct add_dlc_trace
{
      AP_UINT16           opcode;             /* verb operation code         */
      unsigned char       reserv2;            /* reserved                    */
      unsigned char       format;             /* reserved                    */
      AP_UINT16           primary_rc;         /* primary return code         */
      AP_UINT32           secondary_rc;       /* secondary return code       */
      DLC_TRACE_FILTER    filter;             /* resource to be traced       */
} ADD_DLC_TRACE;

typedef struct dlc_trace_filter
{
      unsigned char       resource_type;      /* type of resource            */
      unsigned char       resource_name[8];   /* name of resource            */
      SNA_LFSID           lfsid;              /* session identifier          */
```

```
      unsigned char       message_type        /* type of messages           */
} DLC_TRACE_FILTER;

typedef struct sna_lfsid
{
      union
      {
         AP_UINT16        session_id;
         struct
         {
           unsigned char  sidh;
           unsigned char  sidl;
         } s;
      } uu;
      AP_UINT16           odai;
} SNA_LFSID;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_ADD_DLC_TRACE

filter.resource_type

Specifies the resource to be traced, and optionally the specific message types to be traced for this resource. Possible values are:

AP_ALL_RESOURCES

Set up tracing options for all DLCs, ports, and LSs.

AP_DLC

Set up tracing options for the DLC named in *resource_name*, and for all ports and LSs that use this DLC.

AP_PORT

Set up tracing options for the port named in *resource_name*, and for all LSs that use this port.

AP_LS

Set up tracing options for the LS named in *resource_name*.

AP_PORT_DEFINED_LS

Set up tracing options for the port named in *resource_name*, and for all defined LSs (but not implicit LSs) that use this port.

AP_PORT_IMPLICIT_LS

Set up tracing options for the port named in *resource_name*, and for all implicit LSs (but not defined LSs) that use this port.

filter.resource_name

The name of the DLC, port, or LS for which tracing is being activated. This parameter is reserved if *resource_type* is set to AP_ALL_RESOURCES.

filter.lfsid

The Local Form Session Identifier for a session on the specified LS. This is only valid for *resource_type* AP_LS, and indicates that only messages on this session are to be traced. The structure contains the following three values, which are returned in the SESSION_STATS section of a QUERY_SESSION verb:

filter.lfsid.uu.s.sidh

Session ID high byte.

filter.lfsid.uu.s.sidl

Session ID low byte.

filter.lfsid.odai

Origin Destination Assignor Indicator.

filter.message_type

The type of messages to trace for the specified resource or session. Set this parameter to AP_TRACE_ALL to trace all messages, or specify one or more of the following values (combined using a logical OR):

AP_TRACE_XID XID messages

AP_TRACE_SC	Session Control RUs
AP_TRACE_DFC	Data Flow Control RUs
AP_TRACE_FMD	FMD messages
AP_TRACE_SEGS	Non-BBIU segments that do not contain an RH
AP_TRACE_CTL	Messages other then MUs and XIDs
AP_TRACE_NLP	Trace Network-Layer Protocol messages
AP_TRACE_NC	Trace Network Control messages

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK
secondary_rc	Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	Possible values are:
	AP_INVALID_RESOURCE_TYPE
	The *resource_type* parameter specified a value that was not valid.
	AP_INVALID_MESSAGE_TYPE
	The *message_type* parameter specified a value that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

APING

APING is the APPN version of the “ping” utility; it allows a management application to check the communications path from a local LU to a remote LU in the network.

SNAplus2 APING is implemented using an internally-defined APPC TP. This TP sends data to the partner LU, and optionally receives data from the partner LU. If the TP completes successfully, the APING verb returns information about the time taken to allocate a conversation to the partner LU and to send and receive data.

The application must supply a VCB that is large enough to include a partner TP verification string of the requested size as well as the basic APING VCB structure; the returned data includes this string appended to the end of the basic structure.

This verb is intended for checking the path to an LU on a remote node. Using APING to check communications with a partner LU on the local node will impact the performance of other programs on the local computer, and is not recommended.

This verb must be issued to a running node.

VCB Structure

```
typedef struct aping
{
     AP_UINT16         opcode;           /* verb operation code           */
     unsigned char     reserv2;          /* reserved                      */
     unsigned char     format;           /* reserved                      */
     AP_UINT16         primary_rc;       /* primary return code           */
     AP_UINT32         secondary_rc;     /* secondary return code         */
     unsigned char     lu_name[8];       /* local LU name                 */
     unsigned char     lu_alias[8];      /* local LU alias                */
     AP_UINT32         sense_data;       /* sense data                    */
     unsigned char     plu_alias[8];     /* partner LU alias              */
     unsigned char     mode_name[8];     /* mode name                     */
     unsigned char     tp_name[64];      /* destination TP name           */
     unsigned char     security;         /* security level                */
     unsigned char     reserv3a[3];      /* reserved                      */
     unsigned char     pwd[10];          /* password                      */
     unsigned char     user_id[10];      /* user ID                       */
     AP_UINT16         dlen;             /* length of data to send        */
```

```
     AP_UINT16         consec;            /* number of consecutive sends     */
     unsigned char     fqplu_name[17];    /* fully qualified partner LU name */
     unsigned char     echo;              /* data echo flag                  */
     AP_UINT16         iterations         /* number of iterations            */
     AP_UINT32         alloc_time;        /* time taken for ALLOCATE         */
     AP_UINT32         min_time;          /* minimum send/receive time       */
     AP_UINT32         avg_time;          /* average send/receive time       */
     AP_UINT32         max_time;          /* maximum send/receive time       */
     AP_UINT16         partner_ver_len;   /* size of string to receive       */
     PARTNER_VER_DATA  partner_ver_data;
} APING;

typedef struct partner_ver_data
{
     AP_UINT16         partner_ver_len;   /* size of string to receive      */
}
PARTNER_VER_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_APING

lu_name

LU name of the local LU. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To indicate that the LU is identified by its LU alias instead of its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter.

lu_alias

LU alias of the local LU. This parameter is used only if the *lu_name* field is set to 8 binary zeros, and is ignored otherwise. The alias is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To use the default LU (the LU associated with the CP), set both the *lu_name* and *lu_alias* parameters to 8 binary zeros.

plu_alias

Partner LU alias. This should be the alias of an LU on a remote node; you are not recommended to use APING with a partner LU on the local node.

The alias is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is identified by its fully qualified name instead of its alias, set this parameter to 8 binary zeros and specify the LU name in the *fqplu_name* parameter.

mode_name

Name of the mode used by the LU pair. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with spaces if the name is shorter than 8 characters.

tp_name

Name of the invoked TP (generally set to APINGD). This is a 64-byte string, padded on the right with spaces.

security

Specifies whether conversation security information is required to start the TP. Possible values are:

AP_NONE

No security information is required.

AP_SAME

Security information may be verified by the TP that invoked this TP on behalf of a third TP.

AP_PGM

A user ID and password are required to start the TP.

AP_PGM_STRONG

A password and user ID are required to start the TP. If password substitution is not supported on the session, the aping fails. Otherwise, the password is sent encrypted.

pwd

Password required to access the partner TP; this

parameter is required only if the security parameter is set to AP_PGM. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces if the password is shorter than 10 bytes.

user_id

User ID required to access the partner TP; this parameter is required only if the security parameter is set to AP_SAME or AP_PGM. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces if the user ID is shorter than 10 bytes.

dlen

Length of the data string to be sent to the partner LU. (The NOF API application does not need to provide a data string; the APING TP simply sends a string of zeros of the specified length.)

consec

Number of consecutive data strings sent to the partner LU during each iteration. The APING TP sends this number of data strings, each containing the number of bytes specified by the *dlen* parameter. It then requests either data or a confirmation message from the partner TP, depending on the setting of the *echo* parameter.

fqplu_name

Fully qualified network name for the partner LU. This parameter is used only if the *plu_alias* field is set to 8 binary zeros, and is ignored otherwise. This should be the name of an LU on a remote node; you are not recommended to use APING with a partner LU on the local node.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

echo

Specifies whether the APING TP requests data from the partner LU after sending data to it. Possible values

are:

AP_YES

After sending the specified number of data strings, APING waits to receive data from the partner LU.

AP_NO

After sending the specified number of data strings, APING requests confirmation from the partner LU, but does not receive data.

iterations

Number of times that the APING TP should perform the sequence of sending data to the partner LU and requesting either data or confirmation.

partner_ver_len

Maximum length of the partner TP verification data string which can be received by the NOF API application. The application must supply a VCB large enough to include this string as well as the basic APING VCB structure, because the string will be appended to the returned VCB.

partner_ver_data

When the verb is returned, data up to the length defined by the *partner_ver_len* parameter is appended to the verb.

Returned Parameters: Successful Execution

If the verb executes successfully, APING returns the following parameters:

primary_rc

AP_OK

alloc_time

The time in milliseconds to allocate a conversation to the partner (the time taken for the MC_ALLOCATE verb issued by the APING TP to complete).

min_time

The minimum time in milliseconds required for a data-sending iteration (the shortest measured time for a single iteration of sending data and receiving either data or confirmation). If iterations was set to zero, this parameter is not used.

avg_time

The average time in milliseconds required for a data-sending iteration (the average time for a single iteration of sending data and receiving either data or confirmation). If iterations was set to zero, this parameter is not used.

max_time

The maximum time in milliseconds required for a data-sending iteration (the longest measured time for a single iteration of sending data and receiving either data or confirmation). If iterations was set to zero, this parameter is not used.

partner_ver_len

Length of verification string returned by the partner TP. The string itself is appended to the end of the VCB.

partner_ver_data

Verification string returned by the partner TP. If *partner_ver_len* is zero, then this string is not returned.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined LU name.

AP_BAD_SECURITY

The security parameter was not set to a valid value.

AP_UNKNOWN_PARTNER_MODE

The value specified for *plu_alias*, *fqplu_name*, or *mode_name* did not match any defined partner LU or mode.

AP_BAD_PARTNER_LU_ALIAS

The value specified for *plu_alias* did not match any defined partner LU.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Allocation Failure

If the verb does not execute because SNAplus2 cannot allocate the APPC conversation, SNAplus2 returns the following parameters:

primary_rc AP_ALLOCATION_ERROR

secondary_rc Possible values are:

AP_ALLOCATION_FAILURE_NO_RETRY

The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. Check the *sense_data* parameter and the error log file for more information. Do not attempt to retry the APING verb until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

The conversation could not be allocated because of a temporary condition, such as a link failure. Check the error log file for more information. Retry the APING verb, preferably after a timeout to allow the condition to clear.

AP_SECURITY_NOT_VALID

The user ID or password specified was not accepted by the partner LU.

AP_TP_NAME_NOT_RECOGNIZED

The partner LU does not recognize the specified TP name.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

The remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the APING verb until the cause of the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

The remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a timeout. The reason for the error may be logged on the remote node. Retry the APING verb, preferably after a timeout to allow the condition to clear.

sense_data If the *secondary_rc* parameter is AP_ALLOCATION_FAILURE_NO_RETRY, this parameter contains the SNA sense data associated with the error. For all other *secondary_rc* values, this parameter is reserved.

Returned Parameters: Conversation Failure

If the verb does not execute because the APPC conversation with the partner TP failed, SNAplus2 returns the following parameters:

primary_rc AP_CONV_FAILURE_NO_RETRY

The conversation was terminated because of a permanent condition, such as a session protocol error. Check the error log file to determine the cause of the error. Do not retry the APING verb until the error has been corrected.

primary_rc AP_CONV_FAILURE_RETRY

The conversation was terminated because of a temporary error. Retry the APING verb. If the problem

occurs again, check the error log file to determine the cause of the error.

primary_rc AP_DEALLOC_ABEND

The partner TP deallocated the conversation because of an error condition. The reason for the error may be logged on the remote node.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

CHANGE_SESSION_LIMIT

The CHANGE_SESSION_LIMIT verb requests SNAplus2 to change the session limits for a particular LU-LU-mode combination. Sessions may be activated or deactivated as a result of processing this verb.

This verb must be issued to a running node.

VCB Structure

```
typedef struct change_session_limit
{
   AP_UINT16       opcode;                  /* verb operation code          */
   unsigned char   reserv2;                 /* reserved                     */
   unsigned char   format;                  /* reserved                     */
   AP_UINT16       primary_rc;              /* primary return code          */
   AP_UINT32       secondary_rc;            /* secondary return code        */
   unsigned char   lu_name[8];              /* local LU name                */
   unsigned char   lu_alias[8];             /* local LU alias               */
   unsigned char   plu_alias[8];            /* partner LU alias             */
   unsigned char   fqplu_name[17];          /* fully qualified partner      */
                                            /*  LU name                     */
   unsigned char   reserv3;                 /* reserved                     */
   unsigned char   mode_name[8];            /* mode name                    */
   unsigned char   reserv3a;                /* reserved                     */
   unsigned char   set_negotiable;          /* set max negotiable limit?    */
   AP_UINT16       plu_mode_session_limit;  /* session limit                */
   AP_UINT16       min_conwinners_source;   /* minimum source contention    */
                                            /*  winner sessions             */
   AP_UINT16       min_conwinners_target;   /* minimum target contention    */
                                            /*  winner sessions             */
   AP_UINT16       auto_act;                /* auto activation limit        */
   unsigned char   responsible;             /* who is responsible for       */
                                            /*  deactivating                */
   unsigned char   reserv4[3];              /* reserved                     */
   AP_UINT32       sense_data;              /* sense data                   */
} CHANGE_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_CHANGE_SESSION_LIMIT

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

plu_alias

LU alias of the partner LU.

This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the partner LU is defined by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. This parameter is used only if the *plu_alias* field is set to zeros; it is ignored if *plu_alias* is specified.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Name of the mode to be used by the LUs.

This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

set_negotiable

Specifies whether the maximum negotiable session limit for this mode should be modified. Possible values are:

AP_YES

Use the value specified by *plu_mode_session_limit* as the maximum negotiable session limit for this LU-LU-mode combination.

AP_NO

Leave the maximum negotiable session limit as the value specified for the mode.

plu_mode_session_limit

Requested total session limit for this LU-LU-mode combination: the maximum number of parallel sessions permitted between these two LUs using this mode. Specify a value in the range 1-32,767. This value may be negotiated with the partner LU.

min_conwinners_source

Minimum number of sessions using this mode for which the local LU is the contention winner. Specify a value in the range 0-32,767. The sum of the *min_conwinners_source* and *min_conwinners_target* parameters must not exceed the *plu_mode_session_limit* parameter.

min_conwinners_target

Minimum number of sessions using this mode for which the partner LU is the contention winner. Specify a value in the range 0-32,767. The sum of the *min_conwinners_source* and *min_conwinners_target* parameters must not exceed the *plu_mode_session_limit* parameter.

auto_act

Number of sessions to automatically activate after the session limit is changed. Specify a value in the range 0-32,767. The actual number of automatically activated sessions is the minimum of this value and the negotiated minimum number of contention winner sessions for the local LU. When sessions are deactivated normally (specifying AP_DEACT_NORMAL) below this limit, new sessions are activated up to this limit.

responsible

Indicates whether the local or partner LU is responsible for deactivating sessions after the session limit is changed. Possible values are:

AP_SOURCE	The local LU is responsible.
AP_TARGET	The partner LU is responsible.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Possible values are:

AP_AS_NEGOTIATED

The session limits were changed, but one or more values were negotiated by the partner LU.

AP_AS_SPECIFIED

The session limits were changed as requested, without being negotiated by the partner LU.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_EXCEEDS_MAX_ALLOWED

The *plu_mode_session_limit*, *min_conwinners_source*, *min_conwinners_target*, or *auto_act* parameter was set to a value outside the valid range.

AP_CANT_CHANGE_TO_ZERO

The *plu_mode_session_limit* parameter cannot be set to zero using this verb; use RESET_SESSION_LIMIT instead.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined local LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined local LU name.

AP_INVALID_MODE_NAME

The *mode_name* parameter did not match any defined mode name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name.

AP_INVALID_RESPONSIBLE

The *responsible* parameter was not set to a valid value.

AP_INVALID_SET_NEGOTIABLE

The *set_negotiable* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_MODE_RESET

No sessions are currently active for this LU-LU-mode combination. Use INITIALIZE_SESSION_LIMIT instead of CHANGE_SESSION_LIMIT to specify the limits.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Session Allocation Error

If the verb does not execute because of a session allocation error, SNAplus2 returns the following parameters:

primary_rc AP_ALLOCATION_ERROR

secondary_rc AP_ALLOCATION_FAILURE_NO_RETRY

A session could not be allocated because of a condition that requires corrective action. Check the *sense_data* parameter and any logged messages to determine the reason for the failure, and take any action required. Do not attempt to retry the verb until the condition has been corrected.

sense_data The SNA sense data associated with the allocation failure.

Returned Parameters: CNOS Processing Errors

If the verb does not execute because of an error, SNAplus2 returns the following parameters.

primary_rc AP_CONV_FAILURE_NO_RETRY

The session limits could not be changed because of a condition that requires action (such as a configuration mismatch or a session protocol error). Check the

SNAplus2 log file for information about the error condition, and correct it before retrying this verb.

primary_rc AP_CNOS_PARTNER_LU_REJECT

secondary_rc AP_CNOS_COMMAND_RACE_REJECT

The verb failed because the specified mode was being accessed by another administration program (or internally by the SNAplus2 software) for session activation or deactivation, or for session limit processing. The application should retry the verb, preferably after a timeout to allow the race condition to be cleared.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

CLOSE_FILE

An application uses this verb to release its handle to the domain configuration file, or to the sna.net file,when it has finished issuing NOF verbs to the file. The file which the application wishes to close is identified by the *target_handle* parameter on the call.

The application should always issue CLOSE_FILE for any open file handles before it exits. After the verb completes successfully, the target handle identifying the file is no longer valid.

This verb must be issued to the domain configuration fileor to the sna.net file.

VCB Structure

```
 typedef struct close_file
{
      AP_UINT16          opcode;          /* verb operation code        */
      unsigned char      reserv2          /* reserved                   */
      unsigned char      format;          /* reserved                   */
      AP_UINT16          primary_rc;      /* primary return code        */
      AP_UINT32          secondary_rc;    /* secondary return code      */
 } CLOSE_FILE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_CLOSE_FILE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters:

primary_rc	AP_STATE_CHECK
secondary_rc	AP_VERB_IN_PROGRESS The specified file cannot be released because a previous verb issued for this target handle is still outstanding. All verbs for the target file must be completed before attempting to close the file.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

CONNECT_NODE

An application uses this verb in order to establish communications with a SNAplus2 node (active or inactive). The verb returns a handle identifying the node, which the application can then use on other NOF verbs to indicate the target for the verb. The application should always issue DISCONNECT_NODE for any open node handles before it exits.

The CONNECT_NODE verb can also be issued on a HP-UX client computer as well as on a server; this allows you to use SET_CS_TRACE, QUERY_CS_TRACE, SET_TRACE_FILE, and QUERY_TRACE_FILE to control client-server tracing for the client. See the descriptions of these verbs for more information. No other NOF verbs can be issued to client computers.

VCB Structure

```
typedef struct connect_node
{
      AP_UINT16         opcode;          /* verb operation code            */
      unsigned char     reserv2;         /* reserved                       */
      unsigned char     format;          /* reserved                       */
      AP_UINT16         primary_rc;      /* primary return code            */
      AP_UINT32         secondary_rc;    /* secondary return code          */
      unsigned char     node_type;       /* which node to connect to       */
      unsigned char     node_name[64];   /* name of Node                   */
      AP_UINT32         target_handle;   /* handle for subsequent verbs    */
      unsigned char     node_status;     /* node status                    */
 } CONNECT_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_CONNECT_NODE

node_type To connect to a particular node in order to manage the node's configuration, set this parameter to AP_SPECIFIED_NODE.

To connect to the node currently acting as the central logger, set this parameter to AP_CENTRAL_LOGGER.

This value is required if the application will be issuing the following verbs:

- SET_CENTRAL_LOGGING, QUERY_CENTRAL_LOGGING
- SET_GLOBAL_LOG_TYPE, QUERY_GLOBAL_LOG_TYPE
- SET_LOG_FILE, QUERY_LOG_FILE (if central logging is in use)

node_name Name of the SNAplus2 node to connect to. This parameter is reserved if *node_type* is set to AP_CENTRAL_LOGGER.

If SNAplus2 is running with all components on a single computer, you can set this parameter to all binary zeros; there is no need to specify the node name. Otherwise, setting this parameter to all binary zeros indicates the default local node (on the same SNAplus2 server as the application).

To connect to a HP-UX client in order to control client-server tracing, set this parameter to all binary zeros. The NOF application must be running on the client computer.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

target_handle Returned value for use on subsequent verbs.

node_status Specifies the status of the node. Possible values are:

AP_NDE_STARTING

The node is in the process of being activated.

AP_NDE_STARTED

The node is active.

AP_NDE_STOPPING

The node is in the process of being deactivated.

AP_NDE_STOPPED

The node is not active.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_NODE_NAME

The value that was specified for the *node_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_CONNECTION_NOT_MADE

An error occurred in connecting to the node.

AP_INVALID_VERSION

The application could not connect to the node, because there was a version mismatch between the SNAplus2 software on the computer where the application is running and the computer where the target node is defined. If you are in the process of upgrading the network, so that different computers are running different levels of the SNAplus2 software, nodes running on the back-level software can be managed only by applications running on the back-level

software.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEACTIVATE_CONV_GROUP

The DEACTIVATE_CONV_GROUP verb requests the deactivation of the session corresponding to the specified conversation group. Although this verb is part of the NOF API, it is primarily intended for use by application programmers writing TPs that use the APPC API. The conversation group identifier is returned by the APPC verbs [MC_]ALLOCATE, [MC_]GET_ATTRIBUTES, and RECEIVE_ALLOCATE.

This verb must be issued to a running node.

VCB Structure

```
typedef struct deactivate_conv_group
{
      AP_UINT16        opcode;          /* verb operation code              */
      unsigned char    reserv2;         /* reserved                         */
      unsigned char    format;          /* reserved                         */
      AP_UINT16        primary_rc;      /* primary return code              */
      AP_UINT32        secondary_rc;    /* secondary return code            */
      unsigned char    lu_name[8];      /* local LU name                    */
      unsigned char    lu_alias[8];     /* local LU alias                   */
      AP_UINT32        conv_group_id;   /* conversation group identifier    */
      unsigned char    type;            /* deactivation type                */
      unsigned char    reserv3[3];      /* reserved                         */
      AP_UINT32        sense_data;      /* deactivation sense data          */
} DEACTIVATE_CONV_GROUP;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DEACTIVATE_CONV_GROUP
lu_name	LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.
lu_alias	LU alias of the local LU, as defined to SNAplus2. This

is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

conv_group_id Conversation group identifier for the session to be deactivated.

type Type of deactivation. Possible values are:

AP_DEACT_CLEANUP

Deactivate the session immediately, without waiting for sessions to end.

AP_DEACT_NORMAL

Do not deactivate the session until all conversations using the session have ended.

sense_data If type is set to AP_DEACT_CLEANUP, this parameter specifies the sense data to be used when deactivating the session. Otherwise this parameter is not used.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_DEACT_CG_INVALID_CGID

The *conv_group_id* parameter did not match any valid conversation group ID.

AP_INVALID_CLEANUP_TYPE

The *type* parameter was not set to a valid value.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEACTIVATE_LU_0_TO_3

The DEACTIVATE_LU_0_TO_3 verb requests SNAplus2 to deactivate the session for a particular LU for use with 3270 emulation or LUA (an LU of type 0, 1, 2, or 3) SNAplus2 deactivates the session by sending a TERM_SELF message to the host for the PLU-SLU session.

This verb must be issued to a running node.

VCB Structure

```
typedef struct deactivate_lu_0_to_3
{
      AP_UINT16         opcode;           /* verb operation code          */
      unsigned char     reserv2;
      unsigned char     format;
      AP_UINT16         primary_rc;       /* primary return code          */
      AP_UINT32         secondary_rc;     /* secondary return code        */
      unsigned char     lu_name[8];       /* LU Name                      */
} DEACTIVATE_LU_0_TO_3;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEACTIVATE_LU_9_TO_3

lu_name LU name of the LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters. This return code can also indicate that there was no active session for the specified LU (implying that the session has already been deactivated).

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	Possible values are:
	AP_INVALID_LU_NAME
	The *lu_name* parameter did not match any defined LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEACTIVATE_SESSION

The DEACTIVATE_SESSION verb requests SNAplus2 to deactivate a particular session, or all sessions on a particular mode.

This verb must be issued to a running node.

VCB Structure

```
typedef struct deactivate_session
{
      AP_UINT16         opcode;           /* verb operation code           */
      unsigned char     reserv2;          /* reserved                      */
      unsigned char     format;           /* reserved                      */
      AP_UINT16         primary_rc;       /* primary return code           */
      AP_UINT32         secondary_rc;     /* secondary return code         */
      unsigned char     lu_name[8];       /* local LU name                 */
      unsigned char     lu_alias[8];      /* local LU alias                */
      unsigned char     session_id[8];    /* session identifier            */
      unsigned char     plu_alias[8];     /* partner LU alias              */
      unsigned char     mode_name[8];     /* mode name                     */
      unsigned char     type;             /* deactivation type             */
      unsigned char     reserv3[3];       /* reserved                      */
      AP_UINT32         sense_data;       /* deactivation sense data       */
      unsigned char     fqplu_name[17];   /* fully qualified partner       */
                                          /*  LU name                      */
      unsigned char     reserv4[20];      /* reserved                      */
} DEACTIVATE_SESSION;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEACTIVATE_SESSION

lu_name LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable

characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

session_id 8-byte identifier of the session to deactivate. If this field is set to 8 binary zeros, SNAplus2 deactivates all sessions for the partner LU and mode.

plu_alias LU alias of the partner LU.

This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the partner LU is defined by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros.

mode_name Name of the mode to be used by the LUs.

This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

type Type of deactivation. Possible values are:

AP_DEACT_CLEANUP

Deactivate the session immediately, without waiting for sessions to end.

AP_DEACT_NORMAL

Do not deactivate the session until all conversations using the session have ended.

sense_data If type is set to AP_DEACT_CLEANUP, this parameter specifies the sense data to be used when deactivating the session. Otherwise this parameter is not used.

fqplu_name Fully qualified LU name for the partner LU, as defined to SNAplus2. This parameter is used only if the *plu_alias* field is set to zeros; it is ignored if *plu_alias* is specified.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up

to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters. This return code can also indicate that the session ID did not match the session ID of an active session (implying that the session has already been deactivated).

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CLEANUP_TYPE

The *type* parameter was not set to a valid value.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined LU name.

AP_INVALID_MODE_NAME

The *mode_name* parameter did not match any defined mode name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_3270_DIAG

DEFINE_3270_DIAG specifies the 3270 diagnostics parameters for SNAplus2: the parameters used to record and display response-time data, and the 3270 user alerts that users can send to the host NetView program. It can be used to define 3270 diagnostics parameters for the first time, or to modify an existing definition.

This verb must be issued to the domain configuration file.

VCB Structure

The DEFINE_3270_DIAG verb contains a variable number of `alert_3270_data` structures, each of which defines a 3270 user alert. The structures are included at the end of the `diag_3270_data` structure; the number of these structures is specified by the *`num_alerts`* parameter.

```
typedef struct define_3270_diag
{
    AP_UINT16          opcode;             /* verb operation code           */
    unsigned char      reserv2;            /* reserved                      */
    unsigned char      format;             /* reserved                      */
    AP_UINT16          primary_rc;         /* primary return code           */
    AP_UINT32          secondary_rc;       /* secondary return code         */
    DIAG_3270_DATA     def_data;
} DEFINE_3270_DIAG;


typedef struct diag_3270_data
{
    unsigned char      rtm_overflow;       /* Send RTM data at counter      */
                                           /*  overflow                     */
    unsigned char      rtm_unbind;         /* Send RTM data at UNBIND       */
    unsigned char      rtm_timer_option;   /* RTM timers option             */
    unsigned char      reserv1;            /* reserved                      */
    AP_UINT16          rtm_thresh1;        /* RTM threshold #1              */
    AP_UINT16          rtm_thresh2;        /* RTM threshold #2              */
    AP_UINT16          rtm_thresh3;        /* RTM threshold #3              */
    AP_UINT16          rtm_thresh4;        /* RTM threshold #4              */
    AP_UINT16          num_alerts;         /* Number of user alerts         */
} DIAG_3270_DATA;
```

```
typedef struct alert_3270_data
{
    AP_UINT16          overlay_size;       /* reserved                    */
    unsigned char      description[53];    /* description                 */
    unsigned char      parameter1[33];     /* parameter 1                 */
    unsigned char      parameter2[33];     /* parameter 2                 */
    unsigned char      parameter3[33];     /* parameter 3                 */
} ALERT_3270_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_3270_DIAG

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

def_data.rtm_overflow

Specifies whether to send RTM data to the host every time one of the RTM counters overflows (reaches its maximum value). There is a counter for each of the intervals defined by the *rtm_thresh* values below. Possible values are:

AP_YES

Send RTM data to the host each time a counter overflows.

AP_NO

Do not send RTM data to the host at counter overflow. RTM data may be lost when a counter overflows.

def_data.rtm_unbind

Specifies whether to send RTM data to the host every time a 3270 session is unbound (ends). Possible values are:

AP_YES

Send RTM data to the host each time a session ends.

AP_NO

Do not send RTM data to the host at session end. RTM data may be lost when the session ends.

def_data.rtm_timer_option

Specifies how the host response time is measured. The response time is defined as the interval between the user pressing ENTER or an AID key to send data to the host, and one of the following events:

AP_RTM_SCREEN

The first data from the host reaches the user's screen.

AP_RTM_UNLOCK

The host unlocks the user's keyboard.

AP_RTM_DIRECTION

The host gives the 3270 emulation program direction, so that the user can send more data.

def_data.rtm_thresh1 through *def_data.rtm_thresh4*

Specify the threshold values used to classify response times. These values divide the range of possible response times into five intervals (below *rtm_thresh1*, between *rtm_thresh1* and *rtm_thresh2*, between *rtm_thresh2* and *rtm_thresh3*, between *rtm_thresh3* and *rtm_thresh4*, and above *rtm_thresh4*). SNAplus2 maintains a counter for each of these intervals; each time a host response is received, SNAplus2 determines which of these intervals it falls into, and increments the appropriate counter.

For each threshold, specify a number representing tenths of seconds; for example, the value 25 indicates 2.5 seconds. The values must be in the range 1-1000 (0.1 second-100 seconds), and must be in ascending order (*rtm_thresh2* must be higher than *rtm_thresh1*, *rtm_thresh3* must be higher than *rtm_thresh2*, and so on).

def_data.num_alerts

The number of 3270 user alerts defined for the SNAplus2 system; the range is 0-20.

The host NetView program identifies each alert by a number in the range 1-20; check with the NetView administrator at the host to determine which alert numbers are used and the meaning of each alert. The alert_3270_data structures supplied to this verb are assigned to these numbers in sequence (the first structure is assigned to alert number 1, the second structure to alert number 2, and so on). If the host does not use all the alert numbers in the range, include a blank alert structure (with all parameters set to null strings) for each unused number. For example, if the host uses only alert numbers 10 and 11, set *num_alerts* to 11 and include 9 blank alert structures followed by the structures for alerts 10 and 11.

For each alert, up to the number specified in *num_alerts*, an alert_3270_data structure is required with the following information:

alert_3270_data.description

A text description of the alert. This is an ASCII string of 1-52 characters, followed by a null character.

This description is displayed in the 3270 user interface, to identify the alert to the user. It is not sent to NetView; the NetView operator identifies the alert by its number.

alert_3270_data.parameter1 through *alert_3270_data.parameter3*

Text strings describing any parameters that the user should enter for the alert. Each one is an ASCII string of 1-32 characters, followed by a null character.

Check with the NetView administrator at the host to determine the parameters required for each alert. An alert may not require all three parameters; if so, specify a null string for any parameters that are not required. The user will only be prompted for a parameter if you specify a description for it.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_RTM_THRESHOLD

One or more of the supplied RTM threshold values was not in the permitted range, or the values were not in ascending order.

AP_INVALID_RTM_TIMER_OPTION

The *rtm_timer_option* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

DEFINE_ADJACENT_LEN_NODE

DEFINE_ADJACENT_LEN_NODE adds entries to the node directory database for an adjacent LEN node and its associated LUs, or adds additional LU entries for a previously-defined LEN node.

This verb is equivalent to a series of DEFINE_DIRECTORY_ENTRY verbs for the LEN node and its associated LUs; it provides a fast method of defining the LEN node's configuration with a single verb. To query the directory entries created by this verb, use QUERY_DIRECTORY_ENTRY.

If the verb is issued to an end node, the LEN node's resources are accessible only to that end node.

VCB Structure

```
typedef struct define_adjacent_len_node
{
      AP_UINT16         opcode;             /* verb operation code          */
      unsigned char     reserv2;            /* reserved                     */
      unsigned char     format;             /* reserved                     */
      AP_UINT16         primary_rc;         /* primary return code          */
      AP_UINT32         secondary_rc;       /* secondary return code        */
      unsigned char     cp_name[17];        /* CP name                      */
      unsigned char     description[32];    /* resource description         */
      unsigned char     reserv1[16];        /* reserved                     */
      unsigned char     num_of_lus;         /* number of LUs                */
      unsigned char     wildcard_lus;       /* wildcard LUs                 */
      unsigned char     reserv3[8];         /* reserved                     */
      unsigned char     lu_names[10][8];    /* LU names                     */
} DEFINE_ADJACENT_LEN_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_ADJACENT_LEN_NODE

cp_name The fully qualified name of the CP in the adjacent LEN end node. This should match the name the LEN node sends on its XIDs (if it supports them), and the

adjacent CP name specified on the DEFINE_LS for the link to the LEN node.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

description A null-terminated text string (0-31 characters followed by a null character) describing the adjacent LEN node. This string is for information only; it is stored in the configuration and returned on the QUERY_DIRECTORY_ENTRY verb, but SNAplus2 does not make any other use of it.

num_of_lus The number of LUs to be defined, in the range 0-10. To define an adjacent node with more than 10 LUs, use multiple DEFINE_ADJACENT_LEN_NODE verbs for the same CP name.

wildcard_lus Indicates whether the specified LU names are wildcard entries or explicit LU names. Possible values are:

AP_YES

The specified LU names are wildcard entries.

AP_NO

The specified LU names are explicit entries.

lu_names The names of the LUs being defined on the LEN node. Each name is an 8-byte type-A EBCDIC character string, right-padded with EBCDIC spaces, corresponding to the second part of the fully qualified LU name (the first part of the fully qualified name is defined by the *cp_name* parameter above).

To define the LU associated with the LEN node's control point (the CP LU or default LU), specify the node's fully qualified CP name in the *cp_name* parameter, and include the “network name” part of this name (the 8 characters after the EBCDIC dot) as one of the LU names.

You can specify a wildcard LU name to match multiple LU names, by specifying only the initial characters of

the name. For example, the wildcard LU name "LU" will match "LUNAME" or "LU01" (but will not match "NAMELU"). However, all the LU names specified on a single verb must be of the same type (wildcard or explicit), as defined by the *wildcard_lus* parameter. To add both types of LU names for the same LEN node, use multiple DEFINE_ADJACENT_LEN_NODE verbs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CP_NAME

The *cp_name* parameter contained a character that was not valid.

AP_INVALID_LU_NAME

One or more of the specified LU names contained a character that was not valid.

AP_INVALID_NUM_LUS

The *num_of_lus* parameter was not in the valid range.

AP_INVALID_WILDCARD_NAME

The *wildcard_lus* parameter was set to AP_YES, but one or more of the specified LU names was already defined on a different parent node.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_CP_NAME

The specified CP name is already defined in a directory entry, and is not a LEN node.

AP_INVALID_LU_NAME

One or more of the specified LU names was already defined on a different parent node.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_CN

DEFINE_CN defines a Connection Network (otherwise known as a Virtual Routing Node or VRN). The verb provides the network qualified name of the connection network along with its Transmission Group (TG) characteristics. Also provided is a list of the names of the local ports that can access this connection network.

DEFINE_CN can be used to redefine an existing Connection Network. In particular, new ports can be added to the list of ports which access the connection network by issuing another DEFINE_CN. (Ports can be removed in the same way by issuing the DELETE_CN verb).

This verb is valid only at an end node, and not at a LEN node.

VCB Structure

```
typedef struct define_cn
{
      AP_UINT16          opcode;              /* verb operation code         */
      unsigned char      reserv2;             /* reserved                    */
      unsigned char      format;              /* reserved                    */
      AP_UINT16          primary_rc;          /* primary return code         */
      AP_UINT32          secondary_rc;        /* secondary return code       */
      unsigned char      fqcn_name[17];       /* name of connection network  */
      CN_DEF_DATA        def_data;            /* CN defined data             */
      unsigned char      port_name[8][8];     /* port names                  */
} DEFINE_CN;

typedef struct cn_def_data
{
      unsigned char      description[32];     /* resource description        */
      unsigned char      reserve0[16];        /* reserved                    */
      unsigned char      num_ports;           /* number of ports on CN       */
      unsigned char      reserve1[16];        /* reserved                    */
      TG_DEFINED_CHARS   tg_chars;            /* TG characteristics          */
} CN_DEF_DATA;

typedef struct tg_defined_chars
{
      unsigned char      effect_cap;          /* effective capacity          */
      unsigned char      reserve1[5];         /* reserved                    */
```

```
      unsigned char      connect_cost;        /* connection cost             */
      unsigned char      byte_cost;           /* byte cost                   */
      unsigned char      reserve2;            /* reserved                    */
      unsigned char      security;            /* security                    */
      unsigned char      prop_delay;          /* propagation delay           */
      unsigned char      modem_class;         /* reserved                    */
      unsigned char      user_def_parm_1;     /* user-defined parameter 1    */
      unsigned char      user_def_parm_2;     /* user-defined parameter 2    */
      unsigned char      user_def_parm_3;     /* user-defined parameter 3    */
} TG_DEFINED_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_CN

fqcn_name

Fully qualified name of the connection network. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the connection network. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_CN verb, but SNAplus2 does not make any other use of it.

def_data.num_ports

Number of ports included on this verb; each DEFINE_CN verb can specify up to 8 ports. To define a CN with more than 8 ports, issue multiple DEFINE_CN verbs for the same CN name; the maximum total number of ports on a CN is 239.

def_data.tg_chars.effect_cap

Actual bits per second rate (line speed). The value is

encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

def_data.tg_chars.connect_cost

Cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

def_data.tg_chars.byte_cost

Cost per byte. Valid values are integer values in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

def_data.tg_chars.security

Security level of the network. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

def_data.tg_chars.prop_delay

Propagation delay: the time that a signal takes to travel the length of the link. Specify one of the following values, according to the type of link:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

def_data.tg_chars.user_def_parm_1 through *def_data.tg_chars.user_def_parm_3*

User-defined parameters, which you can use to include other TG characteristics not covered by the above parameters. Each of these parameters must be set to a value in the range 0-255.

port_name

Array of up to eight port names defined on the connection network. Each port name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, and must already have been defined by a DEFINE_PORT verb. Additional ports may be defined on the Connection Network by issuing another DEFINE_CN specifying the new port

names.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_DEF_LINK_INVALID_SECURITY

The *security* parameter was not set to one of the valid values.

AP_EXCEEDS_MAX_ALLOWED

Adding the specified number of ports would exceed the maximum total number of ports on a CN.

AP_INVALID_CN_NAME

The *fqcn_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_NUM_PORTS_SPECIFIED

The *num_ports* parameter was not set to a valid value.

AP_INVALID_PORT_NAME

One or more of the port names specified did not match the name of a defined port.

AP_INVALID_PORT_TYPE

One or more of the specified ports cannot be on a CN because its DLC type is a point-to-point type (such as SDLC) rather than a network type.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all

NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_PORT_ACTIVE

The specified port cannot be modified because it is currently active.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node is a LEN node, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node is a LEN node. This verb is valid only at an end node.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_COS

DEFINE_COS adds a class of service definition or modifies a previously defined COS. The definition specifies TG "rows" and node "rows", which associate a range of node and TG characteristics with weights used for route calculation. The lower the weight the more favorable the route.

VCB Structure

The DEFINE_COS verb contains a variable number of `cos_tg_row` and `cos_node_row` structures; the number of each is specified by the *num_of_node_rows* and *num_of_tg_rows* parameters. The TG rows are included at the end of the main DEFINE_COS structure, in ascending order of weight; they are followed by the node rows, again in ascending order of weight.

```
typedef struct define_cos
{
   AP_UINT16          opcode;                 /* verb operation code        */
   unsigned char      reserv2;                /* reserved                   */
   unsigned char      format;                 /* reserved                   */
   AP_UINT16          primary_rc;             /* primary return code        */
   AP_UINT32          secondary_rc;           /* secondary return code      */
   unsigned char      cos_name[8];            /* class of service name      */
   unsigned char      description[32];        /* resource description       */
   unsigned char      reserv1[16];            /* reserved                   */
   unsigned char      transmission_priority;  /* transmission priority      */
   unsigned char      reserv3[9];             /* reserved                   */
   unsigned char      num_of_node_rows;       /* number of node rows        */
   unsigned char      num_of_tg_rows;         /* number of TG rows          */
} DEFINE_COS;


typedef struct cos_tg_row
{
   TG_DEFINED_CHARS   minimum;                /* minimum                    */
   TG_DEFINED_CHARS   maximum;                /* maximum                    */
   unsigned char      weight;                 /* weight                     */
   unsigned char      reserv1;                /* reserved                   */
} COS_TG_ROW;


typedef struct tg_defined_chars
{
```

```
   unsigned char      effect_cap;             /* effective capacity         */
   unsigned char      reserve1[5];            /* reserved                   */
   unsigned char      connect_cost;           /* cost per connect time      */
   unsigned char      byte_cost;              /* cost per byte              */
   unsigned char      reserve2;               /* reserved                   */
   unsigned char      security;               /* security                   */
   unsigned char      prop_delay;             /* propagation delay          */
   unsigned char      modem_class;            /* reserved                   */
   unsigned char      user_def_parm_1;        /* user defined parameter 1   */
   unsigned char      user_def_parm_2;        /* user defined parameter 2   */
   unsigned char      user_def_parm_3;        /* user defined parameter 3   */
} TG_DEFINED_CHARS;

typedef struct cos_node_row
{
   COS_NODE_STATUS     minimum;               /* minimum                    */
   COS_NODE_STATUS     maximum;               /* maximum                    */

   unsigned char       weight;                /* weight                     */
   unsigned char       reserv1;               /* reserved                   */
} COS_NODE_ROW;

typedef struct cos_node_status
{
   unsigned char       rar;                   /* route additional resistance*/
   unsigned char       status;                /* node status                */
   unsigned char       reserv1[2];            /* reserved                   */
} COS_NODE_STATUS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_COS

cos_name

Class of service name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

description

A null-terminated text string (0-31 characters followed

by a null character) describing the COS. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_COS verb, but SNAplus2 does not make any other use of it.

transmission_priority

Transmission priority. Possible values are:

AP_LOW

AP_MEDIUM

AP_HIGH

AP_NETWORK

num_of_node_rows

Number of node rows which follow the DEFINE_COS VCB (after the TG rows). The maximum is 8.

num_of_tg_rows

Number of TG rows which follow the DEFINE_COS VCB. The maximum is 8.

Each TG row contains a set of minimum TG characteristics, a set of maximum TG characteristics, and a weight. When computing the weights for a TG, its characteristics are checked against the minimum and maximum characteristics defined for each TG row. The TG is then assigned the weight of the first TG row which bounds all the TG's characteristics within the limits specified. If the TG characteristics do not satisfy any of the listed TG rows, the TG is considered unsuitable for this COS, and is assigned an infinite weight. The TG rows must be concatenated in ascending order of weight.

cos_tg_row.minimum.effect_cap

Minimum limit for actual bits per second rate (line speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

cos_tg_row.minimum.connect_cost

Minimum limit for cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

cos_tg_row.minimum.byte_cost

Minimum limit for cost per byte. Valid values are integer values in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

cos_tg_row.minimum.security

Minimum level of security. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

cos_tg_row.minimum.prop_delay

Minimum limits for propagation delay: the time that a signal takes to travel the length of the link. Specify one of the following values, according to the type of link:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

cos_tg_row.minimum.user_def_parm_1 through *cos_tg_row.user_def_parm_3*

Minimum values for user-defined parameters, which you can use to include other TG characteristics not covered by the above parameters. Each of these parameters must be set to a value in the range 0-255.

cos_tg_row.maximum.effect_cap

Maximum limit for actual bits per second rate (line speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

cos_tg_row.maximum.connect_cost

Maximum limit for cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

cos_tg_row.maximum.byte_cost

Maximum limit for cost per byte. Valid values are integer values in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

cos_tg_row.maximum.security

Maximum level of security. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

cos_tg_row.maximum.prop_delay

Maximum limits for propagation delay: the time that a signal takes to travel the length of the link. Specify one of the following values, according to the type of link:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a

LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

cos_tg_row.maximum.user_def_parm_1 through *cos_tg_row.maximum.user_def_parm_3*

Maximum values for user-defined parameters, which you can use to include other TG characteristics not covered by the above parameters. Each of these parameters must be set to a value in the range 0-255.

cos_tg_row.weight

Weight associated with this TG row.

Each node row contains a set of minimum node characteristics, a set of maximum node characteristics, and a weight. When computing the weights for a node, its characteristics are checked against the minimum and maximum characteristics defined for each node row. The node is then assigned the weight of the first node row which bounds all the node's characteristics within the limits specified. If the node characteristics do not satisfy any of the listed node rows, the node is considered unsuitable for this COS, and is assigned an infinite weight. The node rows must be listed in ascending order of weight.

cos_node_row.minimum.rar

Route additional resistance minimum. Values must be in the range 0-255.

cos_node_row.minimum.status

Specifies the minimum congestion status of the node. Possible values are:

AP_UNCONGESTED

The number of ISR sessions is below the *isr_sessions_upper_threshold* value in the node's configuration.

AP_CONGESTED

The number of ISR sessions exceeds the threshold value.

cos_node_row.maximum.rar

Route additional resistance maximum. Values must be in the range 0-255.

cos_node_row.maximum.status

Specifies the maximum congestion status of the node. Possible values are:

AP_UNCONGESTED

The number of ISR sessions is below the *isr_sessions_upper_threshold* value in the node's configuration.

AP_CONGESTED

The number of ISR sessions exceeds the threshold value.

cos_node_row.weight

Weight associated with this node row.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_COS_NAME

The *cos_name* parameter contained a character that was not valid.

AP_INVALID_NUMBER_OF_NODE_ROWS

The *num_of_node_rows* parameter was not in the valid range.

AP_INVALID_NUMBER_OF_TG_ROWS

The *num_of_tg_rows* parameter was not in the valid range.

AP_NODE_ROW_WGT_LESS_THAN_LAST

The node rows were not listed in ascending order of weight.

AP_TG_ROW_WGT_LESS_THAN_LAST

The TG rows were not listed in ascending order of weight.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_COS_TABLE_FULL

You cannot define a new COS because this would exceed the maximum number of COS definitions permitted for the node (specified by the

cos_cache_size parameter on DEFINE_NODE).

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_CPIC_SIDE_INFO

This verb adds or replaces a side information entry. A CPI-C side information entry associates a set of conversation characteristics with a symbolic destination name. If there is already a side information entry with the same symbolic destination name as the one supplied with this verb, it is overwritten with the data supplied to this call.

Note the difference between this verb and the CPI-C function `Set_CPIC_Side_Information`. This verb modifies the domain configuration file, so that it affects all SNAplus2 CPI-C applications. The CPI-C function modifies the application's own copy in memory of the side information table, and does not affect any other CPI-C applications.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct define_cpic_side_info
{
   AP_UINT16               opcode;            /* verb operation code       */
   unsigned char           reserv2;           /* reserved                  */
   unsigned char           format;            /* reserved                  */
   AP_UINT16               primary_rc;        /* primary return code       */
   AP_UINT32               secondary_rc;      /* secondary return code     */
   unsigned char           reserv2a[8];       /* reserved                  */
   unsigned char           sym_dest_name[8];  /* Symbolic destination name */
   CPIC_SIDE_INFO_DEF_DATA  def_data;
} DEFINE_CPIC_SIDE_INFO;


typedef struct cpic_side_info_def_data
{
   unsigned char    description[32];             /* resource description    */
   unsigned char    reserv1[16];                 /* reserved                */
   CPIC_SIDE_INFO   side_info;                   /* CPIC side info          */
   unsigned char    reserved[24];                /* reserved                */

} CPIC_SIDE_INFO_DEF_DATA;


typedef struct cpic_side_info
{
   unsigned char    partner_lu_name[17];         /* Fully qualified         */
```

```
                                          /*  partner LU name        */
   unsigned char    reserved[3];          /* Reserved                */
   AP_UINT32        tp_name_type;         /* TP name type            */
   unsigned char    tp_name[64];          /* TP name                 */
   unsigned char    mode_name[8];         /* Mode name               */
   AP_UINT32        conversation_security_type;  /* Conversation security  */
                                          /*  type                   */
   unsigned char    security_user_id[10];  /* User ID                */
   unsigned char    security_password[10];  /* Password              */
   unsigned char    lu_alias[8];          /* LU alias                */

} CPIC_SIDE_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_CPIC_SIDE_INFO

sym_dest_name

Symbolic destination name which identifies the side information entry. This is an 8-byte ASCII string, padded on the right with spaces if necessary. The name can contain any displayable character.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the side information entry. This string is for information only; it is stored in the configuration file and returned on the QUERY_CPIC_SIDE_INFO verb, but SNAplus2 does not make any other use of it.

def_data.side_info.partner_lu_name

Fully qualified name of the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

def_data.side_info.tp_name_type

The type of the target TP (the valid characters for a TP name are determined by the TP type). Possible values are:

XC_APPLICATION_TP

Application TP. All characters in the TP name must be valid ASCII characters.

XC_SNA_SERVICE_TP

Service TP. All characters, except the first, in the TP name must be valid ASCII characters. The first character must be a hexadecimal digit in the range 0x0-0x3F, excluding 0x0E and 0x0F.

def_data.side_info.tp_name

TP name of the target TP. This is a 64-byte ASCII character string, padded on the right with ASCII spaces.

def_data.side_info.mode_name

Name of the mode used to access the target TP. This is an 8-byte ASCII character string, padded on the right with spaces.

def_data.side_info.conversation_security_type

Specifies whether the target TP uses conversation security. Possible values are:

XC_SECURITY_NONE

The target TP does not use conversation security.

XC_SECURITY_PROGRAM

The target TP uses conversation security. The *security_user_id* and *security_password* parameters specified below will be used to access the target TP.

XC_SECURITY_PROGRAM_STRONG

As for XC_SECURITY_PROGRAM, except that the local node must not send the password across the network in clear text format. (This value is included for compatibility with IBM CPI-C implementations. The

SNAplus2 node cannot provide the appropriate restrictions on sending the password; if a CPI-C application attempts to issue the Allocate call with this value set, the call will fail with a return code indicating that the requested security type is not supported.)

XC_SECURITY_SAME

The target TP uses conversation security, and can accept an "already verified" indicator from the local TP. (This indicates that the local TP was itself invoked by another TP, and has verified the security user ID and password supplied by this TP.) The *security_user_id* parameter specified below will be used to access the target TP; no password is required.

def_data.side_info.security_user_id

User ID used to access the partner TP. This parameter is not required if the *conversation_security_type* parameter is set to XC_SECURITY_NONE.

def_data.side_info.security_password

Password used to access the partner TP. This parameter is required only if the *conversation_security_type* parameter is set to XC_SECURITY_PROGRAM or XC_SECURITY_PROGRAM_STRONG.

def_data.side_info.lu_alias

The alias of the local LU used to communicate with the target TP. This alias is a character string using any locally displayable characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_SYM_DEST_NAME The *sym_dest_name* parameter contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DEFAULT_PU

DEFINE_DEFAULT_PU specifies which PU is the default for handling SNAplus2 management services data. Only one default PU for each node can be defined at any time; a second DEFINE_DEFAULT_PU verb for a different PU name overrides the previous definition.

DEFINE_DEFAULT_PU enables the user to define, redefine, or modify any field of a default PU. This verb also enables the user to delete the default PU, by specifying a null PU name.

If an application issues the MS API verb TRANSFER_MS_DATA without specifying a PU name, then the data is routed to the default PU defined for the local node, and sent on this PU's session with the host SSCP. For more information about TRANSFER_MS_DATA, see the *HP-UX SNAplus2 MS Programmers Guide.*

VCB Structure

```
typedef struct define_default_pu
{
      AP_UINT16           opcode;             /* verb operation code       */
      unsigned char       reserv2;            /* reserved                  */
      unsigned char       format;             /* reserved                  */
      AP_UINT16           primary_rc;         /* primary return code       */

      AP_UINT32           secondary_rc;       /* secondary return code     */
      unsigned char       pu_name[8];         /* PU name                   */
      unsigned char       description[32];    /* resource description      */
      unsigned char       reserv1[16];        /* reserved                  */
      unsigned char       reserv3[16];        /* reserved                  */
} DEFINE_DEFAULT_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_DEFAULT_PU

pu_name Name of the default PU; this must be a PU name defined by a previous DEFINE_LS verb. This is an 8-byte type-A EBCDIC string (starting with a letter),

padded on the right with EBCDIC spaces if necessary.

To delete the default PU, specify all zeros.

description A null-terminated text string (0-31 characters followed by a null character) describing the PU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DEFAULT_PU verb, but SNAplus2 does not make any other use of it.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DEFAULTS

DEFINE_DEFAULTS specifies default parameters used by the node.

VCB Structure

```
typedef struct define_defaults
{
      AP_UINT16           opcode;             /* verb operation code         */
      unsigned char       reserv2;            /* reserved                    */
      unsigned char       format;             /* reserved                    */
      AP_UINT16           primary_rc;         /* primary return code         */
      AP_UINT32           secondary_rc;       /* secondary return code       */
      DEFAULT_CHARS       default_chars;      /* default parameters          */
} DEFINE_DEFAULTS;

typedef struct default_chars
{
      unsigned char       description[32];    /* resource description        */
      unsigned char       reserv2[16];        /* reserved                    */
      unsigned char       mode_name[8];       /* default mode name           */
      unsigned char       implicit_plu_forbidden; /* disallow implicit PLUs? */
      unsigned char       specific_security_codes;/* generic security sense  */
                                              /* codes?                      */
      AP_UINT16           limited_timeout;    /* timeout for limited sessions*/
      unsigned char       reserv[244];        /* reserved                    */
} DEFAULT_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DEFAULTS

default_chars.description

A null-terminated text string (0-31 characters followed by a null character) describing the default parameters. This string is for information only; it is stored in the node's configuration file and returned on the

QUERY_DEFAULTS verb, but SNAplus2 does not make any other use of it.

default_chars.mode_name

Name of the default mode. If an application specifies an unrecognized mode name when attempting to start a session, the parameters from this mode will be used as a default definition for the unrecognized mode.

This must be either a mode defined by a previous DEFINE_MODE verb or one of the SNA-defined modes listed in "Purpose of the NOF API". The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if necessary.

default_chars.implicit_plu_forbidden

Specifies whether SNAplus2 puts implicit definitions in place for unknown partner LUs. Possible values are:

AP_YES

SNAplus2 puts implicit definitions in place for unknown partner LUs.

AP_NO

SNAplus2 does not put implicit definitions in place for unknown partner LUs.

default_chars.specific_security_codes

Specifies whether SNAplus2 uses specific sense codes on a security authentication or authorization failure. Specific sense codes are only returned to those partner LUs which have reported support for them on the session. Possible values are:

AP_YES

SNAplus2 uses specific sense codes.

AP_NO

SNAplus2 does not use specific sense codes.

default_chars.limited_timeout

Specifies the timeout after which free limited-resource conwinner sessions are deactivated. Specify a value in

the range 0-65,535 seconds.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_MODE_NAME

The *mode_name* parameter did not match any defined mode name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DIRECTORY_ENTRY

DEFINE_DIRECTORY_ENTRY defines a new entry in the node directory database. This verb cannot be used to modify existing entries. The verb provides a network qualified resource name along with a resource type (network node, end node, LU or Wildcard).

When defining an adjacent node and its LUs, you are recommended to use DEFINE_ADJACENT_LEN_NODE instead of DEFINE_DIRECTORY_ENTRY. This allows you to define the node and its LUs with a single verb. (DEFINE_DIRECTORY_ENTRY defines only a single entry, so you need to use multiple verbs to define entries for the adjacent node and for its LUs.)

Because the database is hierarchical, each entry includes the name of the parent resource; for an LU the parent resource is the owning Control Point, and for an end node or LEN node it is the network node server. However, when DEFINE_DIRECTORY_ENTRY is used on an end node or LEN node to define an adjacent LEN node resource with which it communicates directly, the entry does not include a parent resource name.

You can specify a “wildcard” LU name to match multiple LU names, by specifying only the initial characters of the name. For example, the wildcard LU name APPN.LU will match APPN.LUNAME or APPN.LU01 (but will not match APPN.NAMELU).

VCB Structure

```
typedef struct define_directory_entry
{
   AP_UINT16        opcode;             /* verb operation code             */
   unsigned char    reserv2;            /* reserved                        */
   unsigned char    format;             /* reserved                        */
   AP_UINT16        primary_rc;         /* primary return code             */
   AP_UINT32        secondary_rc;       /* secondary return code           */
   unsigned char    resource_name[17];  /* network qualified resource name */
   unsigned char    reserv1a;           /* reserved                        */
   AP_UINT16        resource_type;      /* resource type                   */
   unsigned char    description[32];    /* resource description            */
   unsigned char    reserv3[16];        /* reserved                        */
   unsigned char    parent_name[17];    /* fully qualified parent name     */
   unsigned char    reserv1b;           /* reserved                        */
```

```
   AP_UINT16        parent_type;         /* parent's resource type          */
   unsigned char    reserv4[8];          /* reserved                        */
} DEFINE_DIRECTORY_ENTRY;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_DIRECTORY_ENTRY

resource_name Fully qualified name of the resource being registered. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

resource_type Specifies the type of the resource being defined. Possible values are:

AP_ENCP_RESOURCE

End node or LEN node

AP_NNCP_RESOURCE

Network node

AP_LU_RESOURCE

LU

AP_WILDCARD_LU_RESOURCE

Wildcard LU name.

For an LU or wildcard LU, the directory entry for the parent resource (the owning CP) must already be defined.

description A null-terminated text string (0-31 characters followed by a null character) describing the directory entry. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DIRECTORY_ENTRY and QUERY_DIRECTORY_LU verbs, but SNAplus2 does not make any other use of it.

parent_name Fully qualified name of the parent resource; for an LU the parent resource is the owning Control Point, and for an end node or LEN node it is the network node server. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

This parameter should be set to all binary zeros in the following cases:

- When registering a network node CP
- When the verb is being issued to an end node or LEN node to define an adjacent LEN node CP with which the local node communicates directly.

parent_type Specifies the parent type of the resource being defined. Possible values are:

AP_ENCP_RESOURCE

End node (for an LU resource owned by an end node)

AP_NNCP_RESOURCE

Network node (for an LU resource owned by a network node, or for an EN or LEN resource).

Set this parameter to zero when no parent resource name is supplied.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_FQ_OWNING_CP_NAME

The *parent_name* parameter did not match the name of a defined resource.

AP_INVALID_LU_NAME

The *resource_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_RESOURCE_TYPE

The *resource_type* parameter was not set to a valid value.

AP_INVALID_WILDCARD_NAME

The *resource_type* parameter was set to AP_WILDCARD_LU_RESOURCE, but the *resource_name* parameter did not contain a valid wildcard entry.

AP_DUPLICATE

The *resource_name* parameter contained a wildcard entry that has already been defined.

AP_INVALID_RESOURCE_NAME

The *resource_name* parameter specified a node name that clashed with the name of the node to which the verb was issued.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DLC

DEFINE_DLC defines a new DLC. It can also be used to modify the DLC-specific parameters of an existing DLC, if the DLC is not currently active, but other parameters cannot be modified.

VCB Structure

```
typedef struct define_dlc

{
   AP_UINT16       opcode;              /* verb operation code              */
   unsigned char   reserv2;             /* reserved                         */
   unsigned char   format;              /* reserved                         */
   AP_UINT16       primary_rc;          /* primary return code              */
   AP_UINT32       secondary_rc;        /* secondary return code            */
   unsigned char   dlc_name[8];         /* name of DLC                      */
   DLC_DEF_DATA    def_data;            /* DLC defined data                 */
} DEFINE_DLC;

typedef struct dlc_def_data
{
   unsigned char   description[32];      /* resource description            */
   unsigned char   initially_active;    /* is the DLC initially active?    */
   unsigned char   reserv1[15];         /* reserved                         */
   unsigned char   dlc_type;            /* DLC type                         */
   unsigned char   neg_ls_supp;         /* negotiable link station support  */
   unsigned char   port_types;          /* port types supported by DLC type */
   unsigned char   reserv3[11];         /* reserved                         */
   AP_UINT16       dlc_spec_data_len;   /* Length of DLC specific data      */
} DLC_DEF_DATA;
```

DLC-specific data for SDLC:

```
typedef struct sdl_spec_data
{

   V0_MUX_INFO       mux_info;        /* Streams config info                */
   AP_UINT16         mu_credit;       /* amount of credit to allow PC to send*/
   unsigned char     stats_support;   /* activate statistics gathering?     */
   unsigned char     reserve1;        /* reserved                           */
   AP_UINT16         sdh_parms_len;   /* Length of HMOD stub create_parms   */
```

```
   SDH_CREATE_PARMS  sdh_parms;       /* HMOD stub create_parms structure    */
} SDL_SPEC_DATA;


typedef struct sdh_create_parms

{
   AP_UINT16         length;          /* Length of HMOD stub create_parms    */
   AP_UINT16         num_ports;       /* max number of ports DLC can support */
   AP_UINT32         creators_pid;    /* process ID of DLC                   */
   V0_MUX_INFO       mux_info;        /* reserved                            */
} SDH_CREATE_PARMS;
```

DLC-specific data for QLLC:

```
typedef struct vql_dlc_spec_data
{
   V0_MUX_INFO       mux_info;        /* streams config info                 */
} VQL_DLC_SPEC_DATA;
```

DLC-specific data for Token Ring, Ethernet, FDDI:

```
typedef struct vdl_dlc_cfg
{
   V0_MUX_INFO       mux_info;               /* Streams config info         */
} VDL_DLC_CFG;
```

For all DLC types:

```
typedef struct v0_mux_info
{
   AP_UINT16         dlc_type;               /* DLC implementation type     */
   unsigned char     reserve1;               /* reserved                    */
   unsigned char     num_mux_ids;            /* reserved                    */
   AP_UINT32         card_type;              /* type of adapter card        */

   AP_UINT32         adapter_number;         /* DLC adapter number          */
   AP_UINT32         oem_data_length;        /* reserved                    */
   int               mux_ids[5];             /* reserved                    */
} V0_MUX_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DLC

dlc_name

Name of the DLC. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the DLC. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DLC verb, but SNAplus2 does not make any other use of it.

def_data.initially_active

Specifies whether this DLC is automatically started when the node is started. Possible values are:

AP_YES

The DLC is automatically started when the node is started.

AP_NO

The DLC is automatically started only if a port or LS that uses it is defined to be initially active; otherwise it must be started manually.

def_data.dlc_type

Type of the DLC. Possible values are:

AP_SDLC	SDLC
AP_X25	QLLC
AP_TR	Token Ring
AP_ETHERNET	Ethernet
AP_FDDI	FDDI

def_data.neg_ls_supp

Specifies whether the DLC supports negotiable link stations. If *dlc_type* is set to AP_QLLC, this must be

set to AP_YES. Possible values are:

AP_YES

Link stations using this DLC may be negotiable.

AP_NO

Link stations using this DLC must be defined as either primary or secondary; negotiable link stations are not supported.

def_data.port_types

If *dlc_type* is set to AP_ETHERNET/AP_FDDI, set this parameter to AP_PORT_SATF. For all other DLC types, this parameter is reserved.

def_data.dlc_spec_data_len

Length, in bytes, of data specific to the type of the DLC. The DLC-specific data structures should be included at the end of the basic VCB structure.

DLC-specific data for SDLC:

sdl_spec_data.mux_info.dlc_type

Type of the DLC. Set this to AP_IMPL_SDLC_SL

sdl_spec_data.mux_info.card_type

Type of the adapter card.

Possible values are:

AP_CARD_HP_PSI

Single-port PSI card

AP_CARD_HP_ACC

8-port ACC card

AP_CARD_HP_EICON_PCI_SDLC

2-port Eicon PCI card

sdl_spec_data.mux_info.adapter_number

Adapter number used by the DLC. If the server contains more than one SDLC adapter card, specify 0 (zero) for the first card, 1 for the second card, and so on.

Otherwise, set this parameter to `0` (zero).

sdl_spec_data.mu_credit

Specifies the credit value for sending DLC_MUs to the link component. Set this parameter to 4.

sdl_spec_data.stats_support

Specifies whether the DLC collects link statistics information. Possible values are:

AP_YES

The DLC collects link statistics information, which can be examined using QUERY_STATISTICS.

AP_NO

The DLC does not collect link statistics information.

sdl_spec_data.sdh_parms_len

Length, in bytes, of the sdh_parms structure. Set this to `sizeof(SDH_CREATE_PARMS)`.

sdl_spec_data.sdh_parms.length

Length, in bytes, of the sdh_parms structure. Set this to `sizeof(SDH_CREATE_PARMS)`.

sdl_spec_data.sdh_parms.num_ports

The maximum number of ports that this DLC will need to support at a time.

sdl_spec_data.sdh_parms.creators_pid

This parameter is reserved (set it to zero).

DLC-specific data for QLLC:

vql_dlc_spec_data.mux_info.dlc_type

Type of the DLC. Set this to `AP_IMPL_NLI_QLLC`.

vql_dlc_spec_data.mux_info.card_type

Type of the adapter card. Set this to `AP_CARD_QLLC_NLI`.

DLC-specific data for Token Ring, Ethernet, FDDI:

vdl_dlc_cfg.mux_info.dlc_type

Type of the DLC. Possible values are:

`AP_IMPL_TR_DLPI`

Token Ring

`AP_IMPL_ETHER_DLPI`

Ethernet

`AP_IMPL_FDDI_DLPI`

FDDI

`vdl_dlc_cfg.mux_info.card_type`

Type of the adapter card. Possible values are:

`AP_CARD_TOKEN_RING_DLPI`

Token Ring

`AP_CARD_ETHERNET_DLPI`

Ethernet

`AP_CARD_FDDI_DLPI`

FDDI

`vdl_dlc_cfg.mux_info.adapter_number`

Adapter number used by the DLC. If the server contains more than one adapter card for this DLC type, specify `0` for the first card, `1` for the second card, and so on. Otherwise, set this parameter to `0` (zero).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

`primary_rc` `AP_OK`

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

`primary_rc` `AP_PARAMETER_CHECK`

secondary_rc Possible values are:

AP_INVALID_DLC_NAME

The supplied *dlc_name* parameter contained a character that was not valid.

AP_INVALID_DLC_TYPE

The supplied *dlc_type* parameter was not one of the allowed values.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_DLC_ACTIVE

The specified DLC cannot be modified because it is currently active.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DLUR_DEFAULTS

DEFINE_DLUR_DEFAULTS defines a default Dependent LU server (DLUS) and a backup default DLUS; if a default DLUS or backup default DLUS is already defined, the verb overrides the existing definition. The default DLUS name is used by DLUR when it initiates SSCP-PU activation for PUs that do not have an explicitly specified associated DLUS. (To define a PU and its associated DLUS, use DEFINE_INTERNAL_PU .)

The verb can also be used to revoke a default DLUS or backup default DLUS, so that none is defined.

VCB Structure

```
typedef struct define_dlur_defaults
{
      AP_UINT16         opcode;               /* verb operation code        */
      unsigned char     reserv2;              /* reserved                   */
      unsigned char     format;               /* reserved                   */
      AP_UINT16         primary_rc;           /* primary return code        */
      AP_UINT32         secondary_rc;         /* secondary return code      */
      unsigned char     description[32];      /* resource description       */
      unsigned char     reserv1[16];          /* reserved                   */
      unsigned char     dlus_name[17];        /* DLUS name                  */
      unsigned char     bkup_dlus_name[17];   /* Backup DLUS name           */
      unsigned char     reserv3;              /* reserved                   */
      unsigned char     dlus_retry_timeout;   /* retry timeout              */
      unsigned char     dlus_retry_limit;     /* retry limit                */
      unsigned char     reserv4[16];          /* reserved                   */
} DEFINE_DLUR_DEFAULTS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DLUR_DEFAULTS

description

A null-terminated text string (0-31 characters followed

by a null character) describing the DLUR defaults. This string is for information only; it is stored in the node's configuration file, but SNAplus2 does not make any other use of it.

dlus_name

Name of DLUS node which will serve as the default. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To revoke the current default DLUS, so that no default DLUS is defined, set this parameter to 17 binary zeros.

bkup_dlus_name

Name of DLUS node which will serve as the backup default. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To revoke the current backup default DLUS, so that no backup default DLUS is defined, set this parameter to 17 binary zeros.

dlus_retry_timeout

Reactivation timer for contacting a DLUS. If SNAplus2 fails to contact the DLUS, this parameter specifies the time in seconds between retries. Specify a value in the range `0x0001-0xFFFF`.

dlus_retry_limit

Retry count for contacting a DLUS. This parameter is used to specify the number of times SNAplus2 should retry if it fails to contact the DLUS on the first attempt.

Specify a value in the range `0x0001-0xFFFE`, or `0xFFFF` to indicate that SNAplus2 should retry indefinitely until it contacts the DLUS.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DLUS_NAME

The supplied *dlus_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_BKUP_DLUS_NAME

The supplied *dlus_name* parameter contained a character that was not valid or was not in the correct format.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of

primary and secondary return codes that are common to all NOF verbs.

DEFINE_DOMAIN_CONFIG_FILE

DEFINE_DOMAIN_CONFIG_FILE specifies a comment string to be included in the header of the domain configuration file, or modifies an existing comment string.

There is no equivalent verb for a node configuration file, because the header for this file does not contain a comment string; use the description parameter in the DEFINE_NODE verb to include comment information in a node configuration file.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct define_domain_config_file
{
   AP_UINT16            opcode;                 /* verb operation code       */
   unsigned char        reserv2;                /* reserved                  */
   unsigned char        format;                 /* reserved                  */
   AP_UINT16            primary_rc;             /* primary return code       */
   AP_UINT32            secondary_rc;           /* secondary return code     */
   CONFIG_FILE_HEADER   hdr;
} DEFINE_DOMAIN_CONFIG_FILE;


typedef struct config_file_header
{
   AP_UINT16            major_version;          /* reserved                  */
   AP_UINT16            minor_version;          /* reserved                  */
   AP_UINT16            update_release;         /* reserved                  */
   AP_UINT32            revision_level;         /* reserved                  */
   unsigned char        comment[100];           /* optional comment string   */
   unsigned char        updating;               /* reserved                  */
} CONFIG_FILE_HEADER;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_DOMAIN_CONFIG_FILE

hdr.comment An optional comment string to store information about

the file. This is an ASCII string of 0-99 characters, followed by a null character. This parameter is for information only; SNAplus2 returns it on the QUERY_DOMAIN_CONFIG_FILE verb, but does not make any other use of it.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DOWNSTREAM_LU

DEFINE_DOWNSTREAM_LU defines a new downstream LU, and maps it to an upstream host LU or LU pool (defined using DEFINE_LU_0_TO_3 or DEFINE_LU_POOL). This allows the downstream LU to access the host computer using the PU concentration feature of SNAplus2. This verb cannot be used to modify an existing downstream LU.

If you need to activate a downstream LU that is already defined (for example, because the downstream workstation has just been activated), issue the DEFINE_DOWNSTREAM_LU verb for that LU.

VCB Structure

```
typedef struct define_downstream_lu
{
    AP_UINT16           opcode;                 /* verb operation code       */
    unsigned char            reserv2;           /* reserved                  */
    unsigned char            format;            /* reserved                  */
    AP_UINT16           primary_rc;             /* primary return code       */
    AP_UINT32            secondary_rc;          /* secondary return code     */
    unsigned char            dslu_name[8];      /* Downstream LU name        */
    DOWNSTREAM_LU_DEF_DATA   def_data;          /* Defined data              */
} DEFINE_DOWNSTREAM_LU;


typedef struct downstream_lu_def_data
    unsigned char            description[32];   /* resource description      */
    unsigned char            reserv1[16];       /* reserved                  */
    unsigned char            nau_address;       /* downstream LU nau address */
    unsigned char            dspu_name[8];      /* Downstream PU name        */
    unsigned char            host_lu_name[8];   /* Host LU or Pool name      */
    unsigned char            allow_timeout;     /* Allow timeout of host LU? */
    unsigned char            delayed_logon;     /* Allow delayed logon to    */
                                                /*  host LU                  */
    unsigned char            reserv2[6];        /* reserved                  */
} DOWNSTREAM_LU_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DOWNSTREAM_LU

dslu_name

Name of the downstream LU that is being defined. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the downstream LU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DOWNSTREAM_LU verb, but SNAplus2 does not make any other use of it.

def_data.nau_address

Network accessible unit address of the downstream LU. This must be in the range 1-255.

def_data.dspu_name

Name of the downstream PU associated with this LU (as specified on the DEFINE_LS). This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

def_data.host_lu_name

Name of the host LU or host LU pool that the downstream LU will be mapped onto. This is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces.

For PU concentration, the host LU cannot be a dependent LU type 6.2. However, if the downstream LU is LU type 6.2, you can configure the host LU as an LU type 0-3 and specify that the model type for the host LU is unknown.

def_data.allow_timeout

Specifies whether to allow the session between the downstream LU and the upstream LU to timeout if the session is left inactive for the timeout period specified

on the upstream LU definition. Possible values are:

AP_YES

Allow the session this downstream LU has with the upstream LU to timeout.

AP_NO

Do not allow the session this downstream LU has with the upstream LU to timeout.

def_data.delayed_logon

Specifies whether to use delayed logon with this downstream LU (the upstream LU is not activated until the user requests it). Possible values are:

AP_YES

Use delayed logon with this downstream LU; the upstream LU is not activated until the user requests it.

AP_NO

Do not use delayed logon with this downstream LU.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DNST_LU_NAME

The supplied *dslu_name* parameter contained a character that was not valid.

AP_INVALID_NAU_ADDRESS

The supplied NAU address was not in the valid range.

AP_INVALID_ALLOW_TIMEOUT

The supplied *allow_timeout* parameter value was not valid.

AP_INVALID_DELAYED_LOGON

The supplied *delayed_logon* parameter value was not valid.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The specified *dspu_name* parameter was not valid.

AP_PU_NOT_DEFINED

The specified *dspu_name* parameter did not match any defined PU name.

AP_INVALID_PU_TYPE

The PU specified by the *dspu_name* parameter is not a downstream PU that supports PU concentration.

AP_LU_ALREADY_DEFINED

An LU with the specified name has already been defined, and cannot be modified using this verb.

AP_LU_NAU_ADDR_ALREADY_DEFD

An LU with the specified NAU address has already been defined.

AP_INVALID_HOST_LU_NAME

The specified host LU name was not valid.

AP_LU_NAME_POOL_NAME_CLASH

The specified LU name clashes with the name of an existing LU pool.

AP_PU_NOT_ACTIVE

The PU specified by the *dspu_name* parameter is not currently active.

AP_LU_ALREADY_ACTIVATING

An LU with the name specified by the *dslu_name* parameter is currently activating.

AP_LU_DEACTIVATING

An LU with the name specified by the *dslu_name* parameter is currently deactivating.

AP_LU_ALREADY_ACTIVE

An LU with the name specified by the *dslu_name* parameter is already active.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration; this is defined by the *pu_conc_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DOWNSTREAM_LU_RANGE

DEFINE_DOWNSTREAM_LU_RANGE defines a new range of downstream LUs, and maps them to an upstream host LU or LU pool (defined using DEFINE_LU_0_TO_3 or DEFINE_LU_POOL). This allows the downstream LUs to access the host computer using the PU concentration feature of SNAplus2. This verb cannot be used to modify existing downstream LUs.

The supplied parameters to this verb include a base name for the new LUs and the range of NAU addresses. The new LU names are generated by combining the base name with the NAU addresses. For example, a base name of LUNME combined with a NAU range of 11 to 14 would define the LUs LUNME011, LUNME012, LUNME013 and LUNME014.

VCB Structure

```
typedef struct define_downstream_lu_range
{
    AP_UINT16         opcode;              /* verb operation code            */
    unsigned char     reserv2;             /* reserved                       */
    unsigned char     format;              /* reserved                       */
    AP_UINT16         primary_rc;          /* primary return code            */
    AP_UINT32         secondary_rc;        /* secondary return code          */
    unsigned char     dslu_base_name[5];   /* Downstream LU base name        */
    unsigned char     description[32];     /* resource description           */
    unsigned char     reserv1[16];         /* reserved                       */
    unsigned char     min_nau;             /* Minimum NAU address in range   */
    unsigned char     max_nau;             /* Maximum NAU address in range   */
    unsigned char     dspu_name[8];        /* Downstream PU name             */
    unsigned char     host_lu_name[8];     /* Host LU or Pool name           */
    unsigned char     allow_timeout;       /* Allow timeout of host LU?      */
    unsigned char     delayed_logon;       /* Allow delayed logon to host LU */
    unsigned char     reserv4[6];          /* reserved                       */
} DEFINE_DOWNSTREAM_LU_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DOWNSTREAM_LU_RANGE

dslu_base_name

Base name for the names of the new LUs. This is a 5-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the base name is less than 5 characters. SNAplus2 generates the LU name for each LU by appending the 3-digit decimal value of the NAU address to this name.

description

A null-terminated text string (0-31 characters followed by a null character) describing the downstream LUs (the same string is used for each LU in the range). This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DOWNSTREAM_LU verb, but SNAplus2 does not make any other use of it.

min_nau

NAU address of the first LU, in the range 1-255.

max_nau

NAU address of the last LU, in the range 1-255.

dspu_name

Name of the downstream PU (as specified on the DEFINE_LS verb) which the downstream LUs in this range will use. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if necessary.

host_lu_name

Name of host LU or host LU pool that the downstream LUs in the given range will be mapped to. This is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces if necessary.

allow_timeout

Specifies whether to allow the sessions this range of downstream LUs have with the upstream LU to timeout if the session is left inactive for the timeout

period specified on the upstream LU definition. Possible values are:

AP_YES

Allow the sessions this range of downstream LUs have with the upstream LU to timeout.

AP_NO

Do not allow the session this range of downstream LUs have with the upstream LU to timeout.

delayed_logon

Specifies whether to use delayed logon with this range of downstream LUs (the upstream LU is not activated until the user requests it). Possible values are:

AP_YES

Use delayed logon with this range of downstream LUs; the upstream LU is not activated until the user requests it.

AP_NO

Do not use delayed logon with this range of downstream LUs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DNST_LU_NAME

The supplied *dslu_base_name* parameter contained a

character that was not valid.

AP_INVALID_NAU_ADDRESS

One or more of the supplied NAU addresses was not in the valid range.

AP_INVALID_ALLOW_TIMEOUT

The supplied *allow_timeout* parameter value was not valid.

AP_INVALID_DELAYED_LOGON

The supplied *delayed_logon* parameter value was not valid.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The specified *dspu_name* parameter was not valid.

AP_PU_NOT_DEFINED

The specified *dspu_name* parameter did not match any defined PU name.

AP_INVALID_PU_TYPE

The PU specified by the *dspu_name* parameter is not a downstream PU that supports PU concentration.

AP_LU_ALREADY_DEFINED

An LU has already been defined with a name that matches one of the names in the range. The existing LU cannot be modified using this verb.

AP_LU_NAU_ADDR_ALREADY_DEFD

An LU has already been defined with an NAU address that matches one of the addresses in the range.

AP_INVALID_HOST_LU_NAME

The specified host LU name was not valid.

AP_LU_NAME_POOL_NAME_CLASH

One of the LU names in the range clashes with the name of an existing LU pool.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration; this is defined by the *pu_conc_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_DSPU_TEMPLATE

The DEFINE_DSPU_TEMPLATE verb defines a template for the downstream LUs that use the SNAplus2 PU concentration feature. This template is used to define downstream LUs on a group of downstream workstations when a workstation connects over an implicit link (a link not previously defined)..

VCB Structure

```
typedef struct define_dspu_template
{
   AP_UINT16               opcode;           /* verb operation code        */
   unsigned char           reserv2;          /* reserved                   */
   unsigned char           format;           /* reserved                   */
   AP_UINT16               primary_rc;       /* primary return code        */
   AP_UINT32               secondary_rc;     /* secondary return code      */
   unsigned char           template_name[8]; /* Name of template           */
   description             description;      /* resource description       */
   unsigned char           modify_template;  /* Modify existing template?  */
   unsigned char           reserv1[11];      /* reserved                   */
   AP_UINT16               max_instance;     /* Max active template        */
                                             /* instances                  */
   AP_UINT16          num_of_dslu_templates;/* number of DSLU templates*/
}DEFINE_DSPU_TEMPLATE;

typedef struct dslu_template
}
   unsigned char           min_nau;          /* Minimum NAU address in range*/
   unsigned char           max_nau;          /* Maximum NAU address in range*/
   unsigned char           allow_timeout;    /* Allow timeout of host LU?   */
   unsigned char           delayed_logon;    /* Allow delayed logon to host */
                                             /*  LU                         */
   unsigned char           reserv1[8];       /* reserved                    */
   unsigned char           host_lu[8];       /* Host LU or Pool name        */
}DSLU_TEMPLATE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_DSPU_TEMPLATE

template_name

The name of the template for downstream LUs that are present on a group of downstream workstations.

description

Resource description that is returned on the QUERY_DSPU_TEMPLATE verb.

modify_template

Specifies whether this verb should add additional DSLU templates to an existing DSPU template or should replace an existing DSPU template. Possible values are:

AP_MODIFY_DSPU_TEMPLATE

If the named DSPU template does not exist, then it is created. If the named DSPU template does exist, then appended DSLU templates are added to the existing DSPU template.

AP_REPLACE_DSPU_TEMPLATE

A new template is created, overwriting any existing definition.

max_instance

The maximum number of instances of the template that can be active concurrently. When the limit is reached, no new instances are created. Specify a value in the range 0-65,535, where 0 (zero) indicates no limit.

The subrecord dslu_template contains the following parameters:

min_nau

NAU address of the first downstream PU, in the range 1-255.

max_nau

NAU address of the last downstream PU, in the range 1-255.

allow_timeout

Specifies whether to timeout host LUs used by the downstream LU if the session is left inactive for the timeout period specified on the host LU definition. Possible values are:

AP_YES

SNAplus2 is allowed to timeout host LUs used by this downstream LU.

AP_NO

SNAplus2 is not allowed to timeout host LUs used by this downstream LU.

delayed_logon

Specifies whether to delay connecting the downstream LU to the host LU until the first data is received from the downstream LU. Possible values are:

AP_YES

SNAplus2 delays connecting the downstream LU to the host LU. A simulated logon screen is sent to the downstream LU.

AP_NO

SNAplus2 does not delay connecting the downstream LU to the host LU.

host_lu

Name of the host LU or host LU pool that the downstream LU uses. This name is an 8-byte character string.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_TEMPLATE_NAME

The name specified for the *template_name* parameter was not valid.

AP_INVALID_NAU_ADDRESS

The *min_nau* or *max_nau* parameter was not valid.

AP_INVALID_NAU_RANGE

The address specified on the *min_nau* or *max_nau* parameters was not in the valid range.

AP_CLASHING_NAU_RANGE

The range of addresses specified by the *min_nau* parameter through the *max_nau* parameter in a dslu_template subrecord clashes with a range specified by another dslu_template subrecord in the template named by the *template_name* parameter.

AP_INVALID_NUM_DSPU_TEMPLATES

The value specified for the *num_of_dslu_templates* parameter was not in the valid range.

AP_INVALID_ALLOW_TIMEOUT

The value specified for the *allow_timeout* parameter was not valid.

AP_INVALID_DELAYED_LOGON

The value specified for the *delayed_logon* parameter was not valid.

AP_INVALID_MODIFY_TEMPLATE

The value specified for the *modify_template* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_HOST_LU_NAME

The specified *host_lu_name* parameter value was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration; this is defined by the *pu_conc_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_EMULATOR_USER

DEFINE_EMULATOR_USER identifies a user of the SNAplus2 system that can use 3270 emulation or 5250 emulation, and defines the resources available to that user. It can be used to define a new user, to define new sessions for use by an existing user, or to modify the user parameters and session parameters for an existing user. (To delete sessions from an existing user, use DELETE_EMULATOR_USER.)

This verb must be issued to the domain configuration file.

VCB Structure

The DEFINE_EMULATOR_USER verb contains a variable number of `session_def_data` structures; these define the user's sessions and (optionally) the user's remap list. The session structures are included at the end of the `def_data` structure; the number of these structures is specified by the *`num_sessions`* parameter.

```
typedef struct define_emulator_user
{
   AP_UINT16                opcode;           /* verb operation code        */
   unsigned char            reserv2;          /* reserved                   */
   unsigned char            format;           /* reserved                   */
   AP_UINT16                primary_rc;       /* primary return code        */
   AP_UINT32                secondary_rc;     /* secondary return code      */
   unsigned char            user_name[32];    /* 3270 user name             */
   EMULATOR_USER_DEF_DATA   def_data;
} DEFINE_EMULATOR_USER;


typedef struct emulator_user_def_data
{
   unsigned char   description[32];   /* Description - null terminated       */
   unsigned char   reserv3[16];       /* reserved                            */
   unsigned char   style_file[9];     /* Initial 3270 style file name        */
                                      /*  - null terminated                  */
   unsigned char   reserv1[3];        /* reserved                            */
   AP_UINT32       num_sessions;      /* Number of sessions being added      */
   AP_UINT32       max_act_sessions;  /* Max number of active 3270 sessions  */
   unsigned char   view_rtm;          /* Can user view RTM info?             */
   unsigned char   alert_permission;  /* Can user send alerts?               */
   unsigned char   change_lu;         /* Can user change LU/Pool accessed?   */
```

```
   unsigned char   change_style;      /* Can user modify initial style file? */
   unsigned char   user_is_group;     /* Does user_name specify a            */
                                      /*  HP-UX group?                       */
   unsigned char   status;            /* reserved                            */
   unsigned char   reserv2;           /* reserved                            */
} EMULATOR_USER_DEF_DATA;

typedef struct session_def_data
{
   AP_UINT16       sub_overlay_size;  /* reserved                            */
   unsigned char   session_name[8];   /* Long session name                   */
   unsigned char   emulator_type;     /* Emulator type - 3270 or 5250        */
   unsigned char   reserv1;           /* reserved                            */
   unsigned char   description[32];   /* Session description                 */
   unsigned char   reserv2[16];       /* reserved                            */
   union
   {
      SESSION_3270_DEF_DATA   def_data_3270;  /* definition of 3270 session  */
      SESSION_5250_DEF_DATA   def_data_5250;  /* definition of 5250 session  */
   } session_variant;
} SESSION_DEF_DATA;

typedef struct session_3270_def_data
{
   unsigned char   lu_name[8];        /* LU/pool name accessed               */
   unsigned char   session_type;      /* Session type - Model 2-5 or printer */
   unsigned char   model_override;    /* Can the user override the model?    */
} SESSION_3270_DEF_DATA;

typedef struct session_5250_def_data
{

   unsigned char   local_lu_alias[8]; /* Local LU alias                      */
   unsigned char   plu_alias[8];      /* Partner LU alias                    */
   unsigned char   fqplu_name[17];    /* Fully-qualified partner LU name     */
   unsigned char   mode_name[8];      /* Mode name                           */
   unsigned char   session_type;      /* Session type - display or printer   */
} SESSION_5250_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_EMULATOR_USER

sub_overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

user_name

The name of the user or group. This is an ASCII string of 1-32 characters, padded on the right with spaces if the name is shorter than 32 characters, which must be one of the following:

- The HP-UX user ID of a 3270 or 5250 user on the SNAplus2 computer (or, for a Windows Client user, the user name specified in the SNAplus2 client network data file). The *user_is_group* parameter below must be set to AP_NO.
- The HP-UX group ID of a group of 3270 or 5250 user on the SNAplus2 computer (or, for a Windows Client user, the group name specified in the SNAplus2 client network data file). The *user_is_group* parameter below must be set to AP_YES.

To set up a default record, which will be used by any user not explicitly named in the configuration, specify the name <DEFAULT> (in uppercase and including the angle bracket characters), and set *user_is_group* to AP_NO. To restrict use of the emulation program to users and groups that are explicitly named in the configuration, do not include a <DEFAULT> record.

When the user starts the emulation program, SNAplus2 checks the user ID against the emulator user records in the configuration to find the correct record, in the following order:

- If the user ID matches a record for a specific user name, this record is used.

- If the user ID cannot be matched but the user's group ID matches a record for a group, this record is used.
- If neither the user ID nor the group ID can be matched, but there is a <DEFAULT> record defined, this record is used.

def_data.description

An optional text string (0-31 characters followed by a null character). The string is for information only; it is stored in the configuration file and returned on the QUERY_EMULATOR_USER_DEF verb, but SNAplus2 does not make use of it. You can use it to store additional information such as the user's full name, to help distinguish between users.

def_data.style_file

The name of this user's default style file. This is a string of 1-8 characters, followed by a null character. For HP-UX, the actual file name used is the name specified here with the extension `.stu` (3270) or `.stx` (5250) appended to it. For Windows, check the emulator documentation for details of style file naming conventions.

def_data.num_sessions

The number of sessions (3270, 5250, or both) being defined or modified by this verb. Each session must be specified by a `session_def_data` structure following the `def_data` structure.

def_data.max_act_sessions

The maximum number of 3270 sessions that this user can access simultaneously. (For a 5250-only user, this field is ignored.) Use this field as follows:

- To prevent the user from using 3270 emulation, set this field to zero.
- To restrict the number of sessions that the user can use simultaneously, set this field to a value between `1` and `10` (or between `1` and the total number of sessions defined for the user, if this is less than `10`).

- To specify no limit, set this field to `0xFFFFFFFF`.

def_data.view_rtm

Specifies whether the user has permission to view Response Time Monitor (RTM) data. This also determines whether the Last Transaction Time Indicator (LTTI) is displayed on the 3270 status line. (For a 5250-only user, this field is ignored.) Possible values are:

AP_YES

The user can view RTM data; the LTTI is displayed.

AP_NO

The user cannot view RTM data; the LTTI is not displayed.

def_data.alert_permission

Specifies whether the user has permission to send 3270 user alerts. (For a 5250-only user, this field is ignored.) Possible values are:

AP_YES

The user can send alerts.

AP_NO

The user cannot send alerts.

def_data.change_lu

Specifies whether the user has permission to change the mapping of 3270 sessions to LUs, either by remapping them from within the 3270 user interface, or by specifying LU names on the command line when starting the 3270 emulation program. (For a 5250-only user, this field is ignored.) Possible values are:

AP_YES

The user can change the LU/session mapping.

AP_NO

The user cannot change the LU/session mapping.

def_data.change_style

Specifies whether the user has permission to load style files, modify style files, or both. Possible values are:

AP_STYLE_ADVANCED

The user can specify a style file name when starting the emulation program, and can create or modify style files using the emulation program's menu interface.

AP_STYLE_NORMAL

The user can specify a style file name when starting the emulation program, and can create or modify style files using the emulation program's menu interface. The user cannot modify the default style file specified by the *style_file* parameter.

The 3270 emulation program does not distinguish between the two values AP_STYLE_ADVANCED and AP_STYLE_NORMAL; to prevent users from modifying the default style file, set the permissions on this file so that users do not have write access to it.

AP_STYLE_INITIAL

The user can specify a style file name on the command line when starting the emulation program, but cannot load, create, or modify style files from within the menu interface. This option allows you to provide a choice of standard initial style files for a user.

AP_STYLE_RESTRICTED

The user is restricted to the default style file specified by the *style_file* parameter, and cannot create or modify style files.

def_data.user_is_group

Specifies whether the *user_name* parameter identifies a user name or a group name. Possible values are:

AP_YES

The *user_name* parameter identifies a group name. Any user in this group will use this emulator user record unless there is an explicit record for the user's user name.

AP_NO

The *user_name* parameter identifies a user name.

For each session being defined by this verb, up to the number specified in *num_sessions*, a session_def_data structure is required with the following information:

session_def_data.session_name

A name identifying the session. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters. If the name matches an existing session name defined for this user, the information for that session is replaced; otherwise a new session is added.

To define 3270 sessions that are initially displayed in the 3270 emulation program's control interface, specify a name in the range SESS0001-SESS0010. These names correspond to the session numbers 1-10 in the control interface; that is, if you use the names SESS0002, SESS0003, and SESS0005, the control interface will display session numbers 2, 3, and 5 using the information defined for these names. (The names SESS0001-SESS0010 are not displayed in the control interface; they are replaced by the long names defined for these sessions in the 3270 style file.)

3270 sessions that are available for the user to remap to must be given names that are not in the range above. The user will be able to access these sessions only if the *change_lu* parameter (see above) allows the user to remap sessions.

session_def_data.emulator_type

Type of the emulation program that uses this session. Possible values are:

AP_3270_SESSION

Session used for 3270 emulation

AP_5250_SESSION

Session used for 5250 emulation

session_def_data.description

An optional text string (0-31 characters followed by a null character). The string is for information only; it is stored in the configuration file and returned on the QUERY_EMULATOR_USER_DEF verb, but SNAplus2 does not make use of it.

For a 3270 session, the following fields are included:

session_def_data.session_variant.def_data_3270.lu_name

Name of the LU or LU pool that this session uses. This is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces. It must match the name of a type 0-3 LU or LU pool that has already been defined.

If you assign the session to an LU that is in an LU pool, SNAplus2 will try to use this LU when the session is activated. If the LU is not available, SNAplus2 will then attempt to find a free LU from the pool (as though the session had been assigned to the pool).

session_def_data.session_variant.def_data_3270.session_type

Type of the LU or pool (display or printer, and the screen model for display LUs). Possible values are:

AP_3270_DISPLAY_MODEL_2

Display model 2 (80 x 24)

AP_3270_DISPLAY_MODEL_3

Display model 3 (80 x 32)

AP_3270_DISPLAY_MODEL_4

Display model 4 (80 x 43)

AP_3270_DISPLAY_MODEL_5

Display model 5 (132 x 27)

AP_PRINTER

Printer LU. (This value is not valid if *lu_name* specifies an LU pool.)

session_def_data.session_variant.def_data_3270.model_override

Specifies whether the user has permission to change the session to use a different screen model from the one specified. Possible values are:

AP_YES

The user can change the screen model.

AP_NO

The user cannot change the screen model.

For a 5250 session, the following fields are included:

session_def_data.session_variant.def_data_5250.local_lu_alias

LU alias of the local LU that this session uses. This is an 8-byte ASCII string (starting with a letter), padded on the right with spaces. It must match the alias of an APPC local LU that has already been defined.

session_def_data.session_variant.def_data_5250.plu_alias

LU alias of the partner LU that this session uses. This is an 8-byte ASCII string (starting with a letter), padded on the right with spaces. It must match the alias of an APPC partner LU that has already been defined. For more information, see “Defining Partner LUs”.

To indicate that the partner LU is identified by its fully qualified name instead of its LU alias, set this parameter to 8 binary zeros, and specify the fully qualified name in the *fqplu_name* parameter.

session_def_data.session_variant.def_data_5250.fqplu_name

Fully qualified name of the partner LU that this session uses. This parameter is used only if *plu_alias* is set to zeros; it is ignored otherwise. For more information, see “Defining Partner LUs”.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

session_def_data.session_variant.def_data_5250.mode_name

Name of the mode that this session uses. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces. The mode name must match the name of an APPC mode that has already been defined; it is generally set to the SNA-defined mode name QPCSUPP.

session_def_data.session_variant.def_data_5250.session_type

Type of the 5250 session (display or printer). Possible values are:

AP_5250_DISPLAY

Display session

AP_5250_PRINTER

Printer session

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_SESSION_TYPE

The *emulator_type* or *session_type* parameter for one or more sessions was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Defining Partner LUs

This verb allows you to specify the partner LU using either its fully qualified LU name or its LU alias. If you specify the fully qualified name, no additional configuration of the partner LU is required; if you specify the LU alias, you must also use DEFINE_PARTNER_LU to define the fully qualified name corresponding to this alias, so that SNAplus2 can locate the LU when it is required by a session.

If the 5250 emulation program requires LUs to be specified by LU alias rather than by fully qualified name (this applies to many Windows-based 5250 emulation programs that use the WinSNA APIs), you must specify the LU alias on this command, and use DEFINE_PARTNER_LU as above.

DEFINE_FOCAL_POINT

The DEFINE_FOCAL_POINT verb specifies the focal point for a particular Management Services category. When a new focal point is specified, SNAplus2 attempts to establish an implicit primary focal point relationship with the specified focal point by sending an MS_CAPABILITIES request.

VCB Structure

```
typedef struct define_focal_point
{
   AP_UINT16        opcode;             /* verb operation code             */
   unsigned char    reserv2;            /* reserved                        */
   unsigned char    format;             /* reserved                        */
   AP_UINT16        primary_rc;         /* primary return code             */
   AP_UINT32        secondary_rc;       /* secondary return code           */
   unsigned char    reserved;           /* reserved                        */
   unsigned char    ms_category[8];     /* management services category    */
   unsigned char    fp_fqcp_name[17];   /* Fully qualified focal           */
                                        /*  point cp name                  */

   unsigned char    ms_appl_name[8];    /* Focal point application name    */
   unsigned char    description[32];    /* resource description            */
   unsigned char    reserv1[16];        /* reserved                        */

   unsigned char    backup;             /* is focal point a backup         */
   unsigned char    reserv3[16];        /* reserved                        */
} DEFINE_FOCAL_POINT;
```

Supplied Parameters

The application supplies the following parameters:

opcode — AP_DEFINE_FOCAL_POINT

ms_category — Management Services category. This may be either one of the category names specified in the MS Discipline-Specific Application Programs table of *Systems Network Architecture: Management Services Reference* (see "Related Publications"), padded with EBCDIC spaces (0x40), or a user-defined category. A

user-defined category name is an 8-byte type-1134 EBCDIC string, padded with EBCDIC spaces (0x40) if necessary.

fp_fqcp_name Fully qualified control point name of the focal point. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

ms_appl_name Focal point application name. This is normally an EBCDIC string, using type-1134 characters; alternatively, it can be one of the MS Discipline-Specific Application Programs specified in *Systems Network Architecture: Management Services Reference* (see "Related Publications"). The string must be 8 characters long; pad on the right with EBCDIC space characters (0x40) if necessary.

description A null-terminated text string (0-31 characters followed by a null character) describing the focal point. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_FOCAL_POINT verb, but SNAplus2 does not make any other use of it.

backup Indicates whether the specified application is the main focal point for this category, or a backup focal point. Possible values are:

AP_YES

Backup focal point (used only if the main focal point is not available).

AP_NO

Main focal point.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK	The focal point was defined as requested.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CATEGORY_NAME

The supplied category name contained a character that was not valid.

AP_INVALID_FP_NAME

The fully qualified name or the application name was not valid.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Replaced

If the verb does not execute successfully because it is followed by another verb specifying a different focal point, SNAplus2 returns the following parameters.

primary_rc AP_REPLACED

Another DEFINE_FOCAL_POINT was issued to the same node while this verb was outstanding, specifying a different focal point for the same MS category. This verb was abandoned; the node will attempt to contact

the focal point specified by the second verb.

Returned Parameters: Unsuccessful

If the verb does not execute successfully because the focal point relationship cannot be established, SNAplus2 returns the following parameters:

primary_rc AP_UNSUCCESSFUL

secondary_rc Possible values are:

AP_IMPLICIT_REQUEST_REJECTED

The specified focal point rejected the request.

AP_IMPLICIT_REQUEST_FAILED

The node could not send the request to the specified focal point; this may be because the specified control point or application was not found.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_INTERNAL_PU

The DEFINE_INTERNAL_PU verb defines a PU on the local node that is served by DLUR. (To define a downstream PU served by PU concentration or to define a local PU that is directly attached to the host, use DEFINE_LS instead of DEFINE_INTERNAL_PU.)

VCB Structure

```
typedef struct define_internal_pu
{
      AP_UINT16             opcode;              /* verb operation code     */
      unsigned char         reserv2;             /* reserved                */
      unsigned char         format;              /* reserved                */
      AP_UINT16             primary_rc;          /* primary return code     */
      AP_UINT32             secondary_rc;        /* secondary return code   */
      unsigned char         pu_name[8];          /* internal PU name        */
      INTERNAL_PU_DEF_DATA  def_data;            /* defined data            */
} DEFINE_INTERNAL_PU;

typedef struct internal_pu_def_data
{
      unsigned char         description[32];      /* resource description    */
      unsigned char         initially_active;     /* is PU initially active? */
      unsigned char         reserv1[15];          /* reserved                */
      unsigned char         dlus_name[17];        /* DLUS name               */
      unsigned char         bkup_dlus_name[17];   /* backup DLUS name        */
      unsigned char         pu_id[4];             /* PU identifier           */
      AP_UINT16             dlus_retry_timeout;   /* DLUS retry timeout      */
      AP_UINT16             dlus_retry_limit;     /* DLUS retry limit        */
      unsigned char         reserv2[4];           /* reserved                */
} INTERNAL_PU_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_INTERNAL_PU

pu_name

Name of the internal PU that is being defined. This is a type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the internal PU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_DLUR_PU and QUERY_PU verbs, but SNAplus2 does not make any other use of it.

def_data.initially_active

Specifies whether this internal PU is automatically started when the node is started. Possible values are:

AP_YES

The PU is automatically started when the node is started.

AP_NO

The PU is not automatically started; it must be started manually.

def_data.dlus_name

Name of DLUS node which DLUR will use when it initiates SSCP-PU activation. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To indicate that DLUR should use the global default DLUS, set this parameter to 17 binary zeros. In this case, you must also use DEFINE_DLUR_DEFAULTS to define the global default DLUS.

def_data.bkup_dlus_name

Name of DLUS node which will serve as the backup DLUS for this PU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of

up to 8 A-string characters.

To indicate that DLUR should use the global backup default DLUS, set this parameter to 17 binary zeros. In this case, you must also use DEFINE_DLUR_DEFAULTS to define the global backup default DLUS.

def_data.pu_id

PU identifier. This is a 4-byte hexadecimal string, consisting of a block number (3 hexadecimal digits) and a node number (5 hexadecimal digits). The PU ID must match the *pu_id* configured at the host.

def_data.dlus_retry_timeout

Reactivation timer for contacting a DLUS. If SNAplus2 fails to contact the DLUS, this parameter specifies the time in seconds between retries.

Specify a value in the range 0x0001-0xFFFF.

def_data.dlus_retry_limit

The interval in seconds between the second and subsequent attempts to contact the DLUS specified by the *def_data.dlus_name* and *def_data.bkup_dlus_name* parameters. The interval between the first and second attempts is always 1 second. If zero is specified, then the defaults specified using the DEFINE_DLUR_DEFAULTS verb are used. .

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *pu_name* parameter contained a character that was not valid.

AP_INVALID_PU_ID

The *pu_id* parameter contained a character that was not valid.

AP_INVALID_DLUS_NAME

The *dlus_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_BKUP_DLUS_NAME

The *bkup_dlus_name* parameter contained a character that was not valid or was not in the correct format.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_PU_ALREADY_DEFINED

A PU with the specified name has already been defined.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of

primary and secondary return codes that are common to all NOF verbs.

DEFINE_LOCAL_LU

The DEFINE_LOCAL_LU verb defines a new local LU. It can also be used to modify the attach routing data or description of an existing LU (or of the default LU associated with the local node's Control Point), but not any of the other parameters; when modifying an existing LU, all the other parameters must be set to their currently defined values.

VCB Structure

```
typedef struct define_local_lu
{
   AP_UINT16            opcode;           /* verb operation code           */
   unsigned char        reserv2;          /* reserved                      */
   unsigned char        format;           /* reserved                      */
   AP_UINT16            primary_rc;       /* primary return code           */
   AP_UINT32            secondary_rc;     /* secondary return code         */
   unsigned char        lu_name[8];       /* local LU name                 */
   LOCAL_LU_DEF_DATA    def_data;         /* defined data                  */
} DEFINE_LOCAL_LU;

typedef struct local_lu_def_data
{
   unsigned char    description[32];       /* resource description          */
   unsigned char    reserv1;               /* reserved                      *
   unsigned char    security_list_name[14] /* security access list name     */
   unsigned char    lu_alias[8];           /* local LU alias                */
   unsigned char    nau_address;           /* NAU address                   */
   unsigned char    syncpt_support;        /* is Syncpoint supported?       */
   AP_UINT16        lu_session_limit;      /* LU session limit              */
   unsigned char    default_pool;          /* is LU in the pool of default  */
                                           /*  LUs?                         */
   unsigned char    reserv2;               /* reserved                      */
   unsigned char    pu_name[8];            /* PU name                       */
   unsigned char    lu_attributes;         /* LU attributes                 */
   unsigned char    sscp_id[6]             /* SSCP ID                       */
   unsigned char    disable;               /* disable or enable local LU    */
  ROUTING_DATA      attach_routing_data;   /* routing data for incoming     */
                                           /* attaches                      */
} LOCAL_LU_DEF_DATA;

typedef struct routing_data
```

```
{
   unsigned char    sys_name[64];          /* Name of target system for TP   */
   signed long      timeout;               /* timeout value in seconds       */
   unsigned char    back_level;            /* is target system back-level?   */
   unsigned char    reserved[59];          /* reserved                       */
} ROUTING_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_LOCAL_LU

lu_name

Name of the local LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

To modify the attach routing data or description of the default LU associated with the local node's Control Point, set this parameter to 8 binary zeros.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the local LU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_LOCAL_LU verb, but SNAplus2 does not make any other use of it.

def_data.security_list_name

Name of the security access list used by this local LU (defined using the DEFINE_SECURITY_ACCESS_LIST verb). This parameter restricts the use of the LU to only the users named in the specified list. To specify that the LU is available for use by any user, set this parameter to 14 binary zeros.

def_data.lu_alias

Alias of the local LU. This is an 8-byte ASCII string,

using any locally displayable characters, padded on the right to 8 bytes if necessary.

def_data.nau_address

Network accessible unit address of the LU. Specify zero if the LU is an independent LU, or an address in the range 1-255 if the LU is a dependent LU.

def_data.syncpt_support

Specifies whether the LU supports Syncpoint functions. Set this to AP_YES only if you have a Sync Point Manager (SPM) and Conversation Protected Resource Manager (C-PRM) in addition to the standard SNAplus2 product. Possible values are:

AP_YES

Syncpoint is supported.

AP_NO

Syncpoint is not supported.

def_data.lu_session_limit

The maximum total number of sessions (across all modes) supported by the LU.

For a dependent LU, this must be set to 1. For an independent LU, specify zero for no limit, or a value in the range 1-65,535. If you specify an explicit limit, note the following:

- If the LU will be communicating with parallel-session remote LUs, the session limit must include sufficient sessions for CNOS negotiation; a safe minimum is 3, or an additional 2 sessions for each partner LU.
- The LU session limit must be greater than or equal to the sum of the session limits for all modes that the LU will use.

def_data.default_pool

Specifies whether the LU is in the pool of default dependent LUs. For more information, see “Default LUs”. Possible values are:

AP_YES

The LU is in the pool of default LUs, and can be used by applications that do not specify an LU name.

AP_NO

The LU is not in the pool.

If the LU is an independent LU, this parameter is reserved.

def_data.pu_name

Name of the PU which this LU will use, as specified on the DEFINE_LS verb. This field is used only by dependent LUs, and should be set to 8 binary zeros for independent LUs. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if necessary.

def_data.lu_attributes

Identifies additional information about the LU. Possible values are:

AP_NONE

No additional information identified.

AP_DISABLE_PWSUB

Disable password substitution support for the local LU.

def_data.sscp_id

Specifies the ID of the SSCP permitted to activate this LU. This ID is a 6-byte binary string. This parameter is used only by dependent LUs, and is set to all binary zeros if the LU is an independent LU or if the LU can be activated by any SSCP.

def_data.disable

Specifies whether the local LU should be disabled or enabled. Possible values are:

AP_YES

Disable the local LU.

AP_NO

Enable the local LU.

def_data.attach_routing_data.sys_name

The name of the target computer for incoming Allocate requests (requests from a partner TP to start an APPC or CPI-C conversation) that arrive at this local LU.

If the target TP is a broadcast queued TP (that is, servers are informed of its location when it starts, so that they can route incoming Allocate requests to it), or if it always runs on the same SNAplus2 server as the node that owns this LU, set this parameter to binary zeros. Otherwise, set it to the name of the computer where the TP runs.

def_data.attach_routing_data.timeout

The timeout value for dynamic load requests. A request will time out if the invoked TP has not issued a Receive_Allocate verb (APPC), or Accept_Conversation or Accept_Incoming (CPI-C), within this time. Specify the timeout value in seconds, or `-1` to indicate no timeout (dynamic load requests will wait indefinitely).

def_data.attach_routing_data.back_level

Specifies whether the target computer specified by the *sys_name* parameter above is a back-level computer. This parameter is used only when you are in the process of migrating a client-server SNAplus2 system to a later version, so that not all servers and clients on the system are running the same version; it is reserved otherwise.

If both the node that owns this LU and the target system are running the same version of SNAplus2, set this parameter to AP_NO.

If the node that owns this LU is running the newer version of SNAplus2, but the target system is running the older version, set this parameter to AP_YES. When you upgrade the target system later, change this parameter back to AP_NO.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The supplied LU name contained a character that was not valid.

AP_INVALID_NAU_ADDRESS

The supplied NAU address was not in the valid range.

AP_INVALID_SESSION_LIMIT

The supplied session limit was not in the valid range.

AP_INVALID_TIMEOUT

The supplied timeout value was not in the valid range.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The *lu_name* or *lu_alias* parameter contained a character that was not valid.

AP_LU_ALREADY_DEFINED

An LU with this name has already been defined. You cannot use this verb to modify any parameters of an existing LU except the attach routing data.

AP_PU_NOT_DEFINED

The *pu_name* parameter did not match any defined PU name.

AP_LU_ALIAS_ALREADY_USED

An LU with this alias has already been defined. You cannot use this verb to modify any parameters of an existing LU except the attach routing data.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Default LUs

You can set up the configuration of local LUs so that applications do not have to specify an LU name explicitly when starting a conversation; the node will select a suitable default LU for the application to use. The method for doing this depends on whether the applications require dependent or independent LUs, as follows. You cannot provide this facility for both dependent and independent LUs.

- If the applications require dependent LUs, use the *default_pool* parameter on DEFINE_LOCAL_LU for one or more dependent LUs, to specify that they can be used as default LUs. When an application attempts to start a conversation without specifying a local LU name, SNAplus2 will select an unused LU from the pool of LUs defined as default LUs.
- You can define LUs on more than one node as default LUs. An application requesting a default LU may be assigned to any of these LUs as available; there is no requirement for the LU to be on the same computer as the application. However, if you are defining

partner LUs for the applications, these must be defined on all nodes where default LUs are defined (so that the application can contact the correct partner LU using any of the default local LUs).

- If the applications require independent LUs, do not use the *default_pool* parameter to define any local LUs as default LUs. In this case, an application requesting a default LU will be assigned to the LU associated with a local node's CP (this is an independent LU automatically defined by SNAplus2 for each node).

DEFINE_LS

DEFINE_LS is used to define a new link station (LS) or modify an existing one. Before issuing this verb, you must issue the DEFINE_PORT verb to define the port that this LS uses. Link specific data is concatenated to the basic structure.

You cannot use DEFINE_LS to modify the port used by an existing LS; the *port_name* specified on the verb must match the previous definition of the LS. The LS can be modified only if it is not started.

VCB Structure

```
typedef struct define_ls
{
   AP_UINT16       opcode;                /* verb operation code            */
   unsigned char   reserv2;               /* reserved                       */
   unsigned char   format;                /* reserved                       */
   AP_UINT16       primary_rc;            /* primary return code            */
   AP_UINT32       secondary_rc;          /* secondary return code          */
   unsigned char   ls_name[8];            /* name of link station           */
   LS_DEF_DATA     def_data;              /* LS defined data                */
} DEFINE_LS;


typedef struct ls_def_data
{
   unsigned char   description[32];        /* resource description          */
   unsigned char   initially_active;       /* is this LS initially active?  */
   AP_UINT16       reserv2;                /* reserved                      */
   AP_UINT16       react_timer;            /* timer for retrying failed LS  */
   AP_UINT16       react_timer_retry;      /* retry count for failed LS     */
   unsigned char   reserv3[10];            /* reserved                      */
   unsigned char   port_name[8];           /* name of associated port       */
   unsigned char   adj_cp_name[17];        /* adjacent CP name              */
   unsigned char   adj_cp_type;            /* adjacent node type            */
   LINK_ADDRESS    dest_address;           /* destination address           */
   unsigned char   auto_act_supp;          /* auto-activate supported       */
   unsigned char   tg_number;              /* pre-assigned TG number        */
   unsigned char   limited_resource;       /* limited resource              */
   unsigned char   solicit_sscp_sessions;  /* solicit SSCP sessions         */
   unsigned char   pu_name[8];             /* Local PU name (reserved if    */
                                           /*  solicit_sscp_sessions is set */
                                           /*  to AP_NO)                    */
```

```
   unsigned char   disable_remote_act;     /* disable remote activation      */
   unsigned char   dspu_services;          /* services provided for downstream
                                               PU                             */
   unsigned char   dspu_name[8];           /* Downstream PU name (reserved if
                                               dspu_service AP_NONE)          */
   unsigned char   dlus_name[17];          /* DLUS name if dspu_services     */
                                           /*  set to AP_DLUR                */
   unsigned char   bkup_dlus_name[17];     /* Backup DLUS name if            */
                                           /*  dspu_services set to AP_DLUR  */
   unsigned char   hpr_supported;          /* reserved                       */
   unsigned char   hpr_link_lvl_error;     /* reserved                       */
   AP_UINT16       link_deact_timer;       /* link deactivation timer        */
   unsigned char   reserv1;                /* reserved                       */
   unsigned char   default_nn_server;      /* default LS to NN server?       */
   unsigned char   ls_attributes[4];       /* LS attributes                  */
   unsigned char   adj_node_id[4];         /* adjacent node ID               */
   unsigned char   local_node_id[4];       /* local node ID                  */
   unsigned char   cp_cp_sess_support;     /* CP-CP session support          */
   unsigned char   use_default_tg_chars;   /* Use the default tg_chars       */
   TG_DEFINED_CHARS tg_chars;              /* TG characteristics             */
   AP_UINT16       target_pacing_count;    /* target pacing count            */
   AP_UINT16       max_send_btu_size;      /* maximum send BTU size          */
   AP_UINT16       ls_role;                /* link station role              */
   unsigned char   max_ifrm_rcvd;          /* no. before acknowldgement      */
   AP_UINT16       dlus_retry_timeout;     /* seconds to recontact a DLUS    */
   AP_UINT16       dlus_retry_limit;       /* attempts to recontact a DLUS   */
   AP_UINT16       reserv4[30];            /* reserved                       */
   AP_UINT16       link_spec_data_len;     /* length of link specific data   */
} LS_DEF_DATA;


typedef struct tg_defined_chars
{
   unsigned char   effect_cap;             /* effective capacity             */
   unsigned char   reserve1[5];            /* reserved                       */
   unsigned char   connect_cost;           /* connection cost                */
   unsigned char   byte_cost;              /* byte cost                      */
   unsigned char   reserve2;               /* reserved                       */
   unsigned char   security;               /* security                       */
   unsigned char   prop_delay;             /* propagation delay              */
   unsigned char   modem_class;            /* reserved                       */
   unsigned char   user_def_parm_1;        /* user-defined parameter 1       */
   unsigned char   user_def_parm_2;        /* user-defined parameter 2       */
   unsigned char   user_def_parm_3;        /* user-defined parameter 3       */
} TG_DEFINED_CHARS;

typedef struct link_address
```

```
{
   AP_UINT16       reserve1;              /* reserved                      */
   AP_UINT16       length;                /* length                        */
   unsigned char   address[32];           /* address                       */
} LINK_ADDRESS;
```

DLC-specific data for SDLC:

```
typedef struct sdl_link_spec_data
{
   V0_MUX_INFO     mux_info;              /* Streams config info           */
   AP_UINT16       reserve8;              /* reserved                      */
   AP_UINT16       reserve9;              /* reserved                      */
   AP_UINT32       contact_timer;         /* contact timer (fast poll,     */
                                          /*  in ms)                       */
   AP_UINT16       contact_timer_retry;   /* contact timer retry           */
   AP_UINT16       reserve1;              /* reserved                      */
   AP_UINT32       contact_timer2;        /* contact timer (slow poll,     */
                                          /*  in ms)                       */
   AP_UINT16       contact_timer_retry2;  /* contact timer 2 retry         */
   AP_UINT16       reserve2;              /* reserved                      */
   AP_UINT32       disc_timer;            /* disconnect timer (in ms)      */
   AP_UINT16       disc_timer_retry;      /* disconnect timer retry        */
   AP_UINT16       reserve3;              /* reserved                      */
   AP_UINT32       nve_poll_timer;        /* negative poll timer(fast poll) */
   AP_UINT16       nve_poll_timer_retry;  /* negative poll timer retry     */
   AP_UINT16       reserve4;              /* reserved                      */
   AP_UINT32       nve_poll_timer2;       /* negative poll timer(slow poll) */
   AP_UINT16       nve_poll_timer_retry2; /* negative poll timer 2 retry   */
   AP_UINT16       reserve5;              /* reserved                      */
   AP_UINT32       no_resp_timer;         /* No response timer (T1 timer))  */
                                          /*  (in ms)                      */
   AP_UINT16       no_resp_timer_retry;   /* No response timer retry       */
   AP_UINT16       reserve6;              /* reserved                      */
   AP_UINT32       rem_busy_timer;        /* Remote busy timer (in ms)     */
   AP_UINT16       rem_busy_timer_retry;  /* Remote busy timer retry       */
   unsigned char   re_tx_threshold;       /* reserved                      */
   unsigned char   repoll_threshold;      /* reserved                      */
   AP_UINT32       rr_timer;              /* RR turnaround timer (in ms)   */
   unsigned char   group_address;         /* reserved                      */
   unsigned char   poll_frame;            /* Poll frame to use when Primary */
                                          /*  and contact polling secondary */
                                          /*  XID, DISC, SNRM, SNRME, TEST  */
   AP_UINT16       poll_on_iframe;        /* Can LS send poll bit on       */
                                          /*  I-frame                      */
```

```
   AP_UINT16       stub_spec_data_len;     /* length of stub specific data   */
   STUB_SPEC_DATA  stub_spec_data;         /* stub specific data             */
} SDL_LINK_SPEC_DATA;


typedef struct stub_spec_data
{
   int             mux_id;                 /* reserved                       */
   unsigned char   opt1;                   /* options flag 1                 */
   unsigned char   opt2;                   /* options flag 2                 */
   unsigned char   pad[2];                 /* reserved                       */
   AP_UINT32       linesp;                 /* line speed in bps              */
   AP_UINT16       rcv_pool_size;          /* initial number of buffers for  */
                                           /*  rcv pool                      */
   AP_UINT16       poll_wait;              /* seconds between polling HMOD   */
                                           /*  for errors                    */
   AP_UINT16       hmod_data_len;          /* length of dial data string     */
   unsigned char   hmod_data[64];          /* dial data string               */
   unsigned char   x21_sequence[255];      /* reserved                       */
   unsigned char   x21_retry_count;        /* reserved                       */
   AP_UINT16       x21_retry_delay;        /* reserved                       */
   AP_UINT16       v25_tx_delay;           /* reserved                       */
   unsigned char   cdstl;                  /* reserved                       */
   unsigned char   reserve1[3];            /* reserved                       */
} STUB_SPEC_DATA;
```

DLC-specific data for QLLC:

```
typedef struct vql_ls_spec_data
{
   V0_MUX_INFO    mux_info;                /* streams config info            */
   AP_UINT16      reserve1;                /* reserved                       */
   AP_UINT16      reserve2;                /* reserved                       */
   unsigned char  vc_type;                 /* Virtual Circuit type           */
   unsigned char  req_rev_charge;          /* request reverse charge if      */
                                           /*  non-zero                      */
   unsigned char  loc_packet;              /* loc->rem packet                */
                                           /* size 2**locpacket              */
   unsigned char  rem_packet;              /* rem->loc packet                */
                                           /* size 2**rempacket              */
   unsigned char  loc_wsize;               /* loc->rem window size           */
   unsigned char  rem_wsize;               /* rem->loc window size           */
   AP_UINT16      fac_len;                 /* X.25 facilities length         */
   unsigned char  fac[32];                 /* X.25 facilities                */
   AP_UINT16      retry_limit;             /* times to retry send QXID,QSM,  */
                                           /*  QDISC                         */
```

```
   AP_UINT16      retry_timeout;          /* timeout for each of above tries*/
   AP_UINT16      idle_timeout;           /* timeout for no Q msgs during   */
                                          /*  init                          */
   AP_UINT16      pvc_id;                 /* PVC logical channel identifier */
   AP_UINT16      sn_id_len;              /* Length of the subnet identifier*/
   unsigned char  sn_id[4];               /* Subnet identifier              */
   AP_UINT16      cud_len;                /* length of any call user data   */
                                          /*  to send                       */
   unsigned char  cud[16];                /* actual call user data          */
   AP_UINT32      xtras;                  /* reserved                       */
   AP_UINT32      xtra_len;               /* reserved                       */
   unsigned char  rx_thruput_class;       /* reserved                       */
   unsigned char  tx_thruput_class;       /* reserved                       */
   unsigned char  cugo;                   /* reserved                       */
   unsigned char  cug;                    /* reserved                       */
   AP_UINT16      cug_index;              /* reserved                       */
   AP_UINT16      nuid_length;            /* reserved                       */
   unsigned char  nuid_data[109];         /* reserved                       */
   unsigned char  reserve3[2];            /* reserved                       */
   unsigned char  rpoa_count;             /* reserved                       */
   AP_UINT16      rpoa_ids[30];           /* reserved                       */
} VQL_LS_SPEC_DATA;
```

DLC-specific data for Token Ring, Ethernet, FDDI:

```
typedef struct vdl_ls_cfg
{
   V0_MUX_INFO     mux_info;              /* Streams config info            */
   AP_UINT16       reserve1;              /* reserved                       */
   AP_UINT16       reserve2;              /* reserved                       */
   AP_UINT16       test_timeout;          /* TEST timeout value in seconds  */
   AP_UINT16       test_retry_limit;      /* TEST retransmission limit      */
   AP_UINT16       xid_timeout;           /* XID timeout value in seconds   */
   AP_UINT16       xid_retry_limit;       /* XID retransmission limit       */
   AP_UINT16       t1_timeout;            /* T1 timeout value in seconds    */
   AP_UINT16       t1_retry_limit;        /* I-frame retransmission limit   */
} VDL_LS_CFG;
```

Data for all DLC types:

```
typedef struct v0_mux_info
{
   AP_UINT16       dlc_type;              /* DLC implementation type        */
   unsigned char   need_vrfy_fixup;       /* LS needs verification with     */
                                          /*  corresponding DLC             */
   unsigned char   num_mux_ids;           /* reserved                       */
```

```
   AP_UINT32       card_type;              /* reserved                    */
   AP_UINT32       adapter_number;         /* reserved                    */
   AP_UINT32       oem_data_length;        /* reserved                    */
   int             mux_ids[5];             /* reserved                    */
} V0_MUX_INFO;
```

For Token Ring, Ethernet, or FDDI, the *address* parameter in the link_address structure is replaced by the following:

```
typedef struct tr_address
{
   unsigned char   mac_address[6];         /* MAC address                 */
   unsigned char   lsap_address;           /* local SAP address           */
} TR_ADDRESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_LS

ls_name

Name of link station. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the LS. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_LS, QUERY_PU, and QUERY_DOWNSTREAM_PU verbs, but SNAplus2 does not make any other use of it.

def_data.initially_active

Specifies whether this LS is automatically started when the node is started. Possible values are:

AP_YES

The LS is automatically started when the node is started.

AP_NO

The LS is not automatically started; it must be started manually.

def_data.react_timer

Reactivation timer for reactivating a failed LS. If the *react_timer_retry* parameter below is nonzero, to specify that SNAplus2 should retry activating the LS if it fails, this parameter specifies the time in seconds between retries. When the LS fails, or when an attempt to reactivate it fails, SNAplus2 waits for the specified time before retrying the activation. If *react_timer_retry* is zero, this parameter is ignored.

def_data.react_timer_retry

Retry count for reactivating a failed LS. This parameter is used to specify whether SNAplus2 should attempt to reactivate the LS if it fails while in use (or if an attempt to start the LS fails).

Specify zero to indicate that SNAplus2 should not attempt to reactivate the LS, or specify the number of retries to be made. A value of 65,535 indicates that SNAplus2 should retry indefinitely until the LS is activated.

SNAplus2 waits for the time specified by the *react_timer* parameter above between successive retries. If the retry count is reached without successfully reactivating the LS, or if a STOP_LS is issued while SNAplus2 is retrying the activation, no further retries are made; the LS remains inactive unless START_LS is issued for it.

If the *auto_act_supp* parameter is set to AP_YES, the reactivation timer fields are ignored; if the link fails, SNAplus2 does not attempt to reactivate it until the user application that was using the session attempts to restart the session.

def_data.port_name

Name of port associated with this link station. This is an 8-byte ASCII string, padded on the right with

spaces if the name is shorter than 8 bytes, which must match the name of a defined port.

def_data.adj_cp_name

Fully qualified name of the adjacent CP for this LS.

If the *adj_cp_type* parameter below is set to AP_NETWORK_NODE or AP_END_NODE, and preassigned TG numbers are being used, set this parameter to the CP name defined at the adjacent node; if the adjacent node sends a CP name during XID exchange, it will be checked against this value.

If *adj_cp_type* is set to AP_BACK_LEVEL_LEN_NODE, SNAplus2 uses this value only as an identifier; set it to any string (of the format described below) that does not match other CP names defined at this node.

If *adj_cp_type* is set to any other value, or if preassigned TG numbers are not being used, there is no need to specify this parameter; SNAplus2 will check the CP name only if one is specified.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

def_data.adj_cp_type

Adjacent node type.

If the adjacent node is an APPN node, and preassigned TG numbers are not being used, this is normally set to AP_APPN_NODE, indicating that the node type is unknown; SNAplus2 will determine the type during XID exchange.

If preassigned TG numbers are being used, you must specify the node type explicitly. You can also specify it as an additional security check if preassigned TG numbers are not being used. In this case, SNAplus2 will reject a connection attempt from the adjacent node if its node type does not match the one specified here. Use one of the following values:

AP_APPN_NODE

The node type is unknown. SNAplus2 will determine the type during XID exchange.

AP_END_NODE

End node, or up-level LEN node (one that includes the Network Name CV in its XID3).

AP_NETWORK_NODE

Network node.

If the adjacent node is not an APPN node, use one of the following values:

AP_BACK_LEVEL_LEN_NODE

Back-level LEN node (one that does not include the Network Name CV in its XID3).

AP_HOST_XID3

Host node; SNAplus2 should respond to a polling XID from the node with a format 3 XID.

AP_HOST_XID0

Host node; SNAplus2 should respond to a polling XID from the node with a format 0 XID.

AP_DSPU_XID

Downstream PU; SNAplus2 should include XID exchange in link activation. The *dspu_name* and *dspu_services* fields must also be set.

AP_DSPU_NOXID

Downstream PU; SNAplus2 should not include XID exchange in link activation. The *dspu_name* and *dspu_services* fields must also be set.

If you want to run independent LU 6.2 traffic over this LS, you must set the *adj_cp_type* parameter to AP_APPN_NODE, AP_END_NODE, AP_NETWORK_NODE, or AP_BACK_LEVEL_LEN_NODE.

def_data.dest_address.length

Length of the destination address field, as specified in

the following parameter or parameters.

For SDLC:

def_data.dest_address.address

Address of the secondary station on this LS.

- If the port that owns this LS is used only for incoming calls (*out_link_act_lim* on DEFINE_PORT is zero), this parameter is reserved.
- If the port that owns this LS is switched primary and is used for outgoing calls (*port_type* is AP_SWITCHED, *ls_role* is AP_LS_PRI, and *out_link_act_lim* on DEFINE_PORT is nonzero), either set this parameter to 0xFF to accept whatever address is configured at the secondary station, or set it to a 1-byte value in the range 0x01-0xFE which must match the value configured at the secondary station.
- Otherwise, set it to a 1-byte value in the range 0x01-0xFE to identify the link station. If the port is primary multi-drop (*ls_role* on DEFINE_PORT is AP_LS_PRI and *tot_link_act_lim* is greater than 1), this address must be different for each LS on the port.

For QLLC:

def_data.dest_address.address

Address of the destination node for this LS. This parameter is used only for SVC outgoing calls (defined by the *vc_type* parameter in the link-specific data, and by the link activation limit parameters on DEFINE_PORT); it is ignored for incoming calls or for PVC.

The address is a string of 1-14 characters. The address is in X.25 (1980) format; later address formats are not supported.

For Token Ring, Ethernet, FDDI:

def_data.dest_address.mac_address

MAC address of adjacent node.

If the local and adjacent nodes are on LANs of different types (one Token Ring or FDDI, the other Ethernet) connected by a bridge, you will probably need to reverse the bit order of the bytes in the MAC address. For more information, see “Bit Ordering in MAC Addresses”. If the two nodes are on the same LAN, or on LANs of the same type connected by a bridge, no change is required.

def_data.dest_address.lsap_address

Local SAP address of adjacent node.

For all link types:

def_data.auto_act_supp

Specifies whether the link can be activated automatically when required by a session. Possible values are:

AP_YES

The link can be activated automatically.

AP_NO

The link cannot be activated automatically.

If this parameter is set to AP_YES:

- The reactivation timer fields are ignored. If the LS fails, SNAplus2 does not attempt to reactivate it until a dependent LU application that was using the session attempts to restart the session; an LS used by independent LUs will not be reactivated by SNAplus2, and must be restarted manually.
- If the link is to an APPN node, the LS must have a preassigned TG number defined (see the following parameter), and *cp_cp_sess_support* must be set to AP_NO.
- If either the local node or the adjacent node is an end node, the LS must also be defined as auto-activatable at the adjacent node.

def_data.tg_number

Preassigned TG number. This parameter is used only if

the adjacent node is an APPN node (*adj_cp_type* is either AP_NETWORK_NODE or AP_END_NODE); it is ignored otherwise.

This TG number is used to represent the link when the link is activated. The node will not accept any other number from the adjacent node during activation of this link; if the adjacent node is using preassigned TG numbers, the same TG number must be defined by the adjacent node on the adjacent link station.

If the local node is a LEN node, or if the adjacent node is a LEN node and the link is to be auto-activatable, set the TG number to 1. Otherwise, specify a number in the range 1-20, or zero to indicate that the TG number is not preassigned and is negotiated when the link is activated.

If a preassigned TG number is defined, the *adj_cp_name* parameter must also be defined, and the *adj_cp_type* parameter must be set to either AP_END_NODE or AP_NETWORK_NODE.

def_data.limited_resource

Specifies whether this link station is to be deactivated when there are no sessions using the link. Link stations on a nonswitched port cannot be configured as limited resource. Possible values are:

AP_NO

The link is not a limited resource and will not be deactivated automatically.

AP_NO_SESSIONS

The link is a limited resource and will be deactivated automatically when no active sessions are using it.

AP_INACTIVITY

The link is a limited resource and will be deactivated automatically when no active sessions are using it, or when no data has flowed on the link for the time period specified by the *link_deact_timer* field.

A limited resource link station may be configured for

CP-CP session support, by setting this field to AP_NO_SESSIONS and *cp_cp_sess_support* to AP_YES. In this case, if CP-CP sessions are brought up over the link, SNAplus2 will not treat the link as a limited resource (and so will not deactivate it).

def_data.solicit_sscp_sessions

Specifies whether to request the adjacent node to initiate sessions between the SSCP and the local CP and dependent LUs. This parameter is used only if the adjacent node is an APPN node (*adj_cp_type* is either AP_NETWORK_NODE or AP_END_NODE); it is ignored otherwise. If the adjacent node is a host (*adj_cp_type* is either AP_HOST_XID3 or AP_HOST_XID0), SNAplus2 always requests the host to initiate SSCP sessions.

Possible values are:

AP_YES

Request the adjacent node to initiate SSCP sessions.

AP_NO

Do not request the adjacent node to initiate SSCP sessions.

If the adjacent node is an APPN node and *dspu_services* is set to a value other than AP_NONE, this parameter must be set to AP_NO.

def_data.pu_name

Name of the local PU that uses this link. This parameter is used only if *adj_cp_type* is set to AP_HOST_XID3 or AP_HOST_XID0, or if *solicit_sscp_sessions* is set to AP_YES; it is ignored otherwise. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

def_data.disable_remote_act

Specifies whether to prevent activation of the LS by the remote node. Possible values are:

AP_YES

The LS can only be activated by the local node; if the remote node attempts to activate it, SNAplus2 will reject the attempt.

AP_NO

The LS can be activated by the remote node.

def_data.dspu_services

Specifies the services which the local node will provide to the downstream PU across this link. This parameter is used only if the adjacent node is a downstream PU or an APPN node with *solicit_sscp_sessions* set to AP_NO; it is reserved otherwise. Possible values are:

AP_PU_CONCENTRATION

Local node will provide PU concentration for the downstream PU. The local node must be defined to support PU concentration.

AP_DLUR

Local node will provide DLUR services for the downstream PU. The local node must be defined to support DLUR. (Not supported on end node.)

AP_NONE

Local node will provide no services for this downstream PU.

def_data.dspu_name

Name of the downstream PU. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

This parameter is required when both of the following conditions are true; otherwise, it is reserved:

- The *solicit_sscp_sessions* parameter is set to AP_NO
- The *dspu_services* parameter is set to AP_PU_CONCENTRATION

def_data.dlus_name

Name of the DLUS node from which DLUR solicits

SSCP services when the link to the downstream node is activated. This field is reserved if *dspu_services* is not set to AP_DLUR.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To specify the global default DLUS, defined using the DEFINE_DLUR_DEFAULTS verb, set this parameter to 17 binary zeros. If this parameter is set to zeros and there is no global default DLUS, then DLUR will not initiate SSCP contact when the link is activated.

def_data.bkup_dlus_name

Name of the backup DLUS node from which DLUR solicits SSCP services if the node specified by *dlus_name* is not active. This field is reserved if *dspu_services* is not set to AP_DLUR.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To specify the global backup default DLUS, defined using the DEFINE_DLUR_DEFAULTS verb, set this parameter to 17 binary zeros.

def_data.hpr_supported

This parameter is reserved.

def_data.hpr_link_lvl_error

This parameter is reserved.

def_data.link_deact_timer

Limited resource link deactivation timer, in seconds. A limited resource link is automatically deactivated if no data flows over the link for the time specified by this parameter. This parameter is not used if *limited_resource* is set to any value other than

INACTIVITY.

The minimum value is 5; values in the range 1-4 will be interpreted as 5.

The value 0 (zero) indicates that no timeout is used (the link is not deactivated, as if *limited_resource* were set to AP_NO.

def_data.default_nn_server

End node: Specifies whether this is a link supporting CP-CP sessions to a network node that can act as the local node's network node server. When the local node has no CP-CP sessions to a network node server and needs to establish them, it checks this parameter on its defined LSs to find a suitable LS to activate. This allows you to specify which adjacent NNs are suitable to act as the NN server (for example, to avoid using NNs that are accessed by expensive or slow links).

Possible values are:

AP_YES

This link supports CP-CP sessions to a network node that can act as the local node's NN server; the local node can automatically activate this link if it needs to contact an NN server. The *cp_cp_sess_support* parameter must be set to AP_YES.

AP_NO

This link should not be automatically activated in an attempt to contact a network node server.

If the local node is not an end node, this parameter is ignored.

def_data.ls_attributes

This array contains further information about the adjacent node, as described in the following parameters:

def_data.ls_attributes[0]

Host type. Set this to AP_SNA unless you are communicating with a host of one of the other types

listed below. Possible values are:

AP_SNA	Standard SNA host.
AP_FNA	Fujitsu Network Architecture (VTAM-F) host.
AP_HNA	Hitachi Network Architecture host.

def_data.ls_attributes[1]

Network Name CV suppression for a link to a back-level LEN node.

If *adj_cp_type* is set to AP_BACK_LEVEL_LEN_NODE or AP_HOST_XID3, specify whether to suppress inclusion of the Network Name CV in the format 3 XID sent to the LEN node, using one of the following values:

AP_NO

Include the Network Name CV in the XID.

AP_SUPPRESS_CP_NAME

Do not include the Network Name CV.

If *adj_cp_type* is set to any other value, this parameter is ignored.

def_data.adj_node_id

Node ID of adjacent node. This is a 4-byte hexadecimal string, consisting of a block number (three hexadecimal digits) and a node number (five hexadecimal digits). Set it to zeros to disable node ID checking. If this link station is defined on a switched port, the *node_id* must be unique, and there may only be one null *node_id* on each switched port.

def_data.local_node_id

Node ID sent in XIDs on this LS. This is a 4-byte hexadecimal string, consisting of a block number (3 hexadecimal digits) and a node number (5 hexadecimal digits). Set it to zeros to use the node ID specified in the DEFINE_NODE verb.

def_data.cp_cp_sess_support

Specifies whether CP-CP sessions are supported. This

parameter is valid only if the adjacent node is an end node or network node (*adj_cp_type* is AP_NETWORK_NODE, AP_END_NODE, or AP_APPN_NODE); it is ignored otherwise.

Possible values are:

AP_YES

CP-CP sessions are supported. The *solicit_sscp_sessions* parameter must be set to AP_NO.

AP_NO

CP-CP sessions are not supported.

def_data.use_default_tg_chars

Specifies whether the default TG characteristics supplied on the DEFINE_PORT verb should be used. The TG characteristics apply only if the link is to an APPN node; this parameter, and the parameters in the tg_chars structure, are ignored otherwise. Possible values are:

AP_YES

Use the default TG characteristics; ignore the tg_chars structure on this verb.

AP_NO

Use the tg_chars structure on this verb.

def_data.tg_chars.effect_cap

Actual bits per second rate (line speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

def_data.tg_chars.connect_cost

Cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

def_data.tg_chars.byte_cost

Cost per byte. Valid values are integer values in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

def_data.tg_chars.security

Security level of the network. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

def_data.tg_chars.prop_delay

Propagation delay: the time that a signal takes to travel the length of the link. Specify one of the following values, according to the type of link:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

def_data.tg_chars.user_def_parm_1 through *def_data.tg_chars.user_def_parm_3*

User-defined parameters, which you can use to include other TG characteristics not covered by the above parameters. Each of these parameters must be set to a value in the range 1-255.

def_data.target_pacing_count

Numeric value between 1 and 32,767 inclusive indicating the desired pacing window size. (The current version of SNAplus2 does not make use of this value.)

def_data.max_send_btu_size

Maximum BTU size that can be sent from this link station. This value is used to negotiate the maximum BTU size that a pair of link stations can use to communicate with each other. The value includes the length of the TH and RH (total 9 bytes) as well as the RU. Specify a value in the range 265-65,535 (265-4105 for SDLC).

def_data.ls_role

Link station role. This is normally set to AP_USE_PORT_DEFAULTS, specifying that the LS role is to be taken from the definition of the port that owns this LS.

If you need to override the port's LS role for an individual LS, specify one of the following values:

AP_LS_PRI	Primary
AP_LS_SEC	Secondary
AP_LS_NEG	Negotiable

def_data.max_ifrm_rcvd

The maximum number of I-frames that can be received by this link station before an acknowledgment is sent. Specify a value in the range 0-127. If 0 is specified, the value from DEFINE_QLLC_PORT DEFINE_SDLC_PORTis used.

def_data.dlus_retry_timeout

Reactivation timer for contacting a DLUS. If SNAplus2 fails to contact the DLUS, this parameter specifies the time in seconds between retries.

Specify a value in the range 0x0001-0xFFFF.

def_data.dlus_retry_limit

The interval in seconds between the second and subsequent attempts to contact the DLUS specified by the *dlus_name* and *bkup_dlus_name* parameters. Specify a value in the range 0x0001-0xFFFE, or specify 0xFFFF to indicate that SNAplus2 should retry indefinitely until it contacts the DLUS. The interval between the first and second attempts is always 1 second. If zero is specified, then the defaults specified using the DEFINE_DLUR_DEFAULTS verb are used. This parameter is ignored if the *dspu_services* parameter is not set to AP_DLUR.

def_data.link_spec_data_len

Length of the link-specific data. The data should be concatenated to the basic structure.

Link-specific data for SDLC:

mux_info.dlc_type

Type of the DLC. Set this to AP_IMPL_SDLC_SL

contact_timer

Timeout required before a SNRM or XID is retransmitted in the event of nonacknowledgment (used for primary SDLC only). This value must be greater than the no response (T1) timeout value *no_resp_timer* described below. The timer is specified in milliseconds.

This timer is also used for special pre-activation polling.

contact_timer_retry

Number of times transmission and retransmission of a contact frame (such as SNRM) is allowed using the normal poll timer before SNAplus2 changes to the slow poll timer.

A value of `0xFFFF` indicates an unlimited retry count. A value of `0x0001` indicates that the switch to the slow poll should be made after the first timer expiry.

contact_timer2

Slow poll contact timer in milliseconds. When the contact timer retry count expires, SNAplus2 continues to poll using this timer. This prevents leased (multi-drop) links from being flooded by poll frames for absent stations.

contact_timer_retry2

The slow poll contact timer retry limit value corresponds to the number of times transmission and retransmission of a contact frame (such as SNRM) is allowed on the slow cycle before an outage message is sent to the DLC user.

A value of `0xFFFF` indicates an unlimited retry count. A value of `0x0001` indicates that an outage should be generated after the first slow poll timer expiry.

disc_timer

Timeout required before a DISC is retransmitted in the event of nonacknowledgment (used for primary SDLC only). The timer is specified in milliseconds.

disc_timer_retry

The Disconnect timer retry limit value corresponds to the number of times transmission and retransmission of a DISC is allowed.

A value of `0xFFFF` indicates an unlimited retry count. A value of `0x0001` indicates that an outage should be generated after the first timer expiry.

nve_poll_timer

Timeout required before an adjacent secondary station (which has previously been removed from the polling list because it has no data to send) is reinserted into the polling list. The timer is specified in milliseconds.

nve_poll_timer_retry

Number of times a station is removed from the polling list on the normal poll timer before SNAplus2 switches to using the slow poll timer.

A value of `0xFFFF` indicates an unlimited retry count. A value of `0x0001` indicates that the switch to the slow poll should be made after the first timer expiry.

nve_poll_timer2

The slow negative poll timer in milliseconds. When the negative poll timer retry count expires, SNAplus2 continues to poll using this timer. This prevents leased (multi-drop) links from being flooded by poll frames for idle stations.

nve_poll_timer_retry2

Number of times a station is removed from the polling list on the slow poll cycle before an outage message is sent to the HLS. Specify a value in the range `1-65,535`. This value is normally set to `0xFFFF`, indicating infinite retry.

no_resp_timer

The maximum time a primary station waits (after having sent a frame with a poll bit) for a response frame before trying to poll another station. This timer is restarted when a frame without the F-bit is received

and stopped only when a frame with an F-bit is received. The timeout should be set to a value not less than twice the transmission time for the longest I-frame plus adjacent station frame processing time.

The timer is specified in milliseconds.

no_resp_timer_retry

Number of times an adjacent secondary station is seen to fail to respond before the primary sends an outage message to the DLC user.

A value of `0xFFFF` indicates an unlimited retry count. A value of `0x0001` indicates that an outage should be generated after the first timer expiry.

rem_busy_timer

Time allowed for an adjacent secondary station to be in an RNR condition. This is used in conjunction with the retry limit value to provide the overall time before an outage message is sent. The timer is specified in milliseconds.

rem_busy_timer_retry

Used in conjunction with the timeout value to provide the overall timeout before an outage message is sent.

A value of `0xFFFF` is used to indicate an unlimited retry count. A value of `0x0001` indicates that an outage should be generated after the first timer expiry.

rr_timer

The time in milliseconds to wait before turning the poll bit around when the SDLC component has no work to do.

This field may be set to zero, or to a nonzero value if the immediate turn-around causes hardware problems on the link. Higher values can also be used to optimize link usage, because often the higher-level software will generate data in response to the data contained in an I-frame carrying the poll bit; the pause allows the data to be received and processed.

poll_frame

The frame to use for pre-activation polling. This is normally XID, indicating that polling is in the control of the DLC user. However, when SNAplus2 is primary talking to an old secondary implementation, it may be necessary to poll using some other frame. Possible values are: `XID`, `DISC`, `SNRM`, `SNRME`, `TEST`.

`poll_on_iframe`

Specifies whether this link station is permitted to send the poll bit on an I-frame. This allows SNAplus2 to work with certain SDLC implementations which do not handle receipt of I-frames carrying the poll bit.

Possible values are:

`AP_YES`

This link station is allowed to send the poll bit on an I-frame.

`AP_NO`

This link station is not allowed to send the poll bit on an I-frame.

`stub_spec_data_len`

Length of the Stub specific data that follows. Set this to size of (STUB_SPEC_DATA).

`stub_spec_data`

Stub specific data. These fields are used only for switched outgoing links. The values specified in this structure override those defined in the Stub specific data for the port that owns this LS; where fields in this structure are shown as reserved, the values from the port are used instead. For switched incoming links or leased links, the parameters defined in the Stub specific data for the port (not for the LS) are used.

The structure contains the following fields:

`stub_spec_data.opt1`

HMOD port options flag 1. Set the appropriate bits of this field as follows (bit 7 is the most significant bit):

`bit 6` Use NRZI (NRZ if not set)

bit 4 Line is full-duplex (half-duplex if not set)

The remaining bits are reserved.

stub_spec_data.opt2

Reserved.

stub_spec_data.linesp

The line speed for the line used on this port. For example, 2400 (0x00000960) for a 2400 baud line. Valid values are in the range 600-38,400 baud. The exact meaning of this parameter depends on the value set on the *physical_link* parameter.

- If *physical_link* is set to SDLC_PL_X21, then the *stub_spec_data.linesp* parameter is ignored.
- If *physical_link* is set to SDLC_PL_V25 or SDLC_PL_SMART_MODEM, then the value of the *stub_spec_data.linesp* parameter is the speed at which the dial string is sent to the modem.
  - — If *physical_link* is set to any other value, then the value of the *stub_spec_data.linesp* parameter is the speed of data transfer, only valid if external clocking is specified.

stub_spec_data.rcv_pool_size

Reserved.

stub_spec_data.poll_wait

Reserved.

stub_spec_data.hmod_data_len

Length of the dial data string that follows (in the *hmod_data* parameter). If no dial data is specified, set this parameter to zero.

stub_spec_data.hmod_data

Dial data for outgoing calls. This parameter applies only to switched links; it is reserved if the port associated with this LS is defined to be nonswitched.

This is an ASCII string, specifying the dial string to be passed to the modem to initiate the call. Support for dial data depends on the SDLC adapter and modem that you are using; if they do not support dial data, set this parameter to a null string.

Link-specific data for QLLC:

mux_info.dlc_type

Type of the DLC. Set this to AP_IMPL_NLI_QLLC.

vc_type

The Virtual Circuit type of the LS. Possible values are:

VQL_SVC	Switched Virtual Circuit
VQL_PVC	Permanent Virtual Circuit

If you define both SVC and PVC LSs between the same local node and remote node, unpredictable results may occur if the SVC LS is started first (since it may not be possible to match the incoming call to the correct LS). To avoid these problems, ensure that PVC LSs are activated before any SVC LSs between the same pair of nodes.

req_rev_charge

Specifies whether X.25 should request reverse charging when attempting to contact the remote system using this LS. Possible values are:

AP_YES

Request reverse charging.

AP_NO

Do not request reverse charging. If the X.25 network does not support facilities negotiation, this parameter must be set to AP_NO.

If the X.25 network does not support facilities negotiation, this parameter must be set to AP_NO (also see the *fac* parameter below).

loc_packet

Packet size used for sending data from the local station

to the remote station. The actual size used is 2 to the power of *loc_packet*; for example, setting *loc_packet* to 7 indicates a packet size of 2 to the power of 7, which is 128. To indicate using the default packet size for the network, set this parameter to zero. Check with the administrator of your X.25 network to determine the correct value to use.

rem_packet

Packet size used for receiving data from the remote station. The actual size used is 2 to the power of *rem_packet* (as for *loc_packet* above). To indicate using the default packet size for the network, set this parameter to zero. Check with the administrator of your X.25 network to determine the correct value to use.

loc_wsize

Window size used for sending data from the local station to the remote station. Specify a value in the range 1-7, or zero to indicate using the default window size for the network. Check with the administrator of your X.25 network to determine the correct value to use.

rem_wsize

Window size used for receiving data from the remote station. Specify a value in the range 1-7, or zero to indicate using the default window size for the network. Check with the administrator of your X.25 network to determine the correct value to use.

fac_len

Length of the additional X.25 facilities data that follows (in the *fac* parameter). If no additional data is required, specify zero. If the X.25 network does not support facilities negotiation, specify zero and see the *fac* parameter below for more information.

fac

If the X.25 provider software is configured to allow flow control negotiation, SNAplus2 includes information

about reverse charging options, packet sizes, and window sizes in the facilities data sent to the remote system. If your X.25 network or the remote system requires any other facilities data in addition to the above parameters, specify it using this parameter. Check with the administrator of your X.25 network, or the administrator of the remote system, to determine what to specify in this parameter.

If the X.25 network does not support facilities negotiation, or if you do not want to use it, ensure that the X.25 provider software is configured not to allow flow control negotiation. In this case, the parameters *req_rev_charge*-*rem_wsize* above must all be set to use the network's default values, and *fac_len* must be set to zero.

retry_limit

Number of times to retry sending a QXID, QSM, or QDISC message if no response is received within the time specified by *retry_timeout* below.

Range is 1-255. If *vc_type* above is set to VQL_PVC, this parameter is ignored for QXID messages; XID sending is retried indefinitely (so that an initially active LS can wait indefinitely for the remote station to become active).

retry_timeout

Timeout in seconds for QXID, QSM, or QDISC messages. A message will be retried (up to the number of times specified by *retry_limit* above) if no response is received within this time. Range is 1-255.

idle_timeout

Timeout in seconds used to detect a completely idle line. This value is used during connect processing for SVCs when the local station is secondary and waiting for XIDs. If no message is received in this time, SNAplus2 assumes that the remote station has failed.

Range is 1-255. This field is ignored if *vc_type* above is set to VQL_PVC.

pvc_id

PVC identifier. Set this to a decimal number to identify which PVC (from the range of PVCs defined for your X.25 provider software) is to be used for this LS. This field is reserved if *vc_type* above is set to VQL_SVC.

sn_id_len

Length of the subnet identifier that follows (in the *sn_id* parameter). Set this parameter to 1 for a single-port card, or 3 for a multi-port card, to match the length of the *sn_id* parameter as described below.

sn_id

Subnet identifier: this parameter identifies the physical connection to the X.25 network.

Set it to a character string that matches the final character(s) of the X.25 driver name, as follows:

- If you have more than one X.25 adapter card installed in the HP-UX computer, set the first byte of this parameter to the ASCII character 0 (zero) to use the first card, the ASCII character 1 (one) to use the second, and so on.
- If you have only one X.25 adapter card, set the first byte of this parameter to the ASCII character 0 (zero) .
- If the card is a multi-port card, set the second byte to the ASCII character p, and the third byte to the ASCII character 1 (one) to indicate the first port on the card, the ASCII character 2 to indicate the second, and so on. For example, 0p2 indicates the second port on the first (multi-port) card.

The parameter must be set to the ASCII character value 0 or 1, not to the numeric value zero or one.

cud_len

Length of the Call User Data that follows (in the *cud* parameter).

cud

Call User Data: this parameter identifies the protocol to be used over the underlying X.25 virtual circuit. For most implementations, this should be set to a single hex byte `0xC3`, indicating QLLC. Some remote systems may require additional bytes; check with the System Administrator of the remote system.

DLC-specific data for Token Ring, Ethernet, FDDI:

`mux_info.dlc_type`

Type of the DLC.

Possible values are:

`AP_IMPL_TR_DLPI`

Token Ring

`AP_IMPL_ETHER_DLPI`

Ethernet

`AP_IMPL_FDDI_DLPI`

FDDI

`test_timeout`

Timeout required before a TEST frame is retransmitted when trying to contact a remote station. The timer is specified in milliseconds. Higher values may be needed if the remote station is on a separate Token Ring connected by a bridge.

`test_retry_limit`

The TEST retry limit value corresponds to the number of times transmission and retransmission of a TEST frame is allowed. This count includes the initial transmission; that is, a value of `1` indicates “transmit once but do not retry”. Higher values may be needed if the remote station is on a separate Token Ring connected by a bridge.

`xid_timeout`

Timeout required before an XID is retransmitted when trying to contact a remote station. The timer is specified in milliseconds. Higher values may be needed

if the remote station is on a separate Token Ring connected by a bridge.

xid_retry_limit

The XID retry limit value corresponds to the number of times transmission and retransmission of an XID is allowed. This count does not include the initial transmission; that is, a value of 1 indicates "transmit once and then retry once". Higher values may be needed if the remote station is on a separate Token Ring connected by a bridge.

t1_timeout

Timeout required before an I-frame is retransmitted if no response is received. The timer is specified in milliseconds. Higher values may be needed if the remote station is on a separate Token Ring connected by a bridge.

Some systems may provide alternative methods of configuring the LLC2 stack, in which case this parameter may be ignored; check with your SNAplus2 supplier.

t1_retry_limit

The T1 retry limit value corresponds to the number of times transmission and retransmission of an I-frame is allowed. The minimum is 1. Higher values may be needed if the remote station is on a separate Token Ring connected by a bridge.

Some systems may provide alternative methods of configuring the LLC2 stack, in which case this parameter may be ignored; check with your SNAplus2 supplier.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_CANT_MODIFY_PORT_NAME

The *ls_name* parameter matched the name of an existing LS, but the *port_name* parameter did not match the existing definition. You cannot modify the port name when changing the definition of an existing LS.

AP_DEF_LINK_INVALID_SECURITY

The *tg_chars.security* parameter was not set to a valid value.

AP_INVALID_AUTO_ACT_SUPP

The *auto_act_supp* parameter was not set to a valid value, or was set to AP_YES when *cp_cp_sess_support* was also set to AP_YES.

AP_INVALID_CP_NAME

The *adj_cp_name* parameter contained a character that was not valid, was not in the correct format, or was not specified when required.

AP_INVALID_LIMITED_RESOURCE

The *limited_resource* parameter was not set to a valid value.

AP_INVALID_LINK_NAME

The *ls_name* parameter contained a character that was not valid.

AP_INVALID_LS_ROLE

The *ls_role* parameter was not set to a valid value.

AP_INVALID_NODE_TYPE

The *adj_cp_type* parameter was not set to a valid

value.

AP_INVALID_PORT_NAME

The *port_name* parameter did not match the name of any defined port.

AP_INVALID_PU_NAME

The *pu_name* parameter did not match the name of any defined PU, or was set when not expected.

AP_INVALID_DSPU_NAME

The *dspu_name* parameter did not match the name of any defined PU, or was set when not expected.

AP_INVALID_DSPU_SERVICES

The *dspu_services* parameter was not set to a valid value, or was set when not expected.

AP_INVALID_DSPU_NAME

A reserved parameter was set to a nonzero value.

AP_INVALID_DSPU_SERVICES

A reserved parameter was set to a nonzero value.

AP_INVALID_SOLICIT_SSCP_SESS

The *solicit_sscp_sess* parameter was not set to a valid value.

AP_INVALID_TARGET_PACING_CNT

The *target_pacing_count* parameter was not set to a valid value.

AP_INVALID_DLUS_NAME

The *dlus_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_BKUP_DLUS_NAME

The *bkup_dlus_name* parameter contained a character that was not valid or was not in the correct format.

AP_HPR_NOT_SUPPORTED

The *hpr_supported* parameter was set to AP_YES, but

the node does not support HPR.

AP_INVALID_TG_NUMBER

The TG number supplied was not in the valid range.

AP_MISSING_CP_NAME

A TG number was defined, but no CP name was supplied.

AP_MISSING_CP_TYPE

A TG number was defined, but no CP type was supplied.

AP_MISSING_TG_NUMBER

The link was defined to be auto-activated, but no TG number was supplied.

AP_PARALLEL_TGS_NOT_SUPPORTED

This node cannot support more than one LS defined between it and the same adjacent node.

AP_INVALID_DLUS_RETRY_LIMIT

The value specified for *dlus_retry_limit* was not valid.

AP_INVALID_DLUS_RETRY_TIMEOUT

The value specified for *dlus_retry_timeout* was not valid.

AP_INVALID_LS_ROLE

The value specified for the *ls_role* parameter is not valid.

AP_INVALID_BTU_SIZE

The value specified for the *max_send_btu_size* parameter was not valid.

AP_INVALID_MAX_IFRM_RCVD

The value specified for the *max_ifrm_rcvd* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all

NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_DUPLICATE_CP_NAME

A link to the CP name specified in the *adj_cp_name* parameter has already been defined.

AP_DUPLICATE_DEST_ADDR

A link to the destination address specified in the *address* parameter has already been defined.

AP_DUPLICATE_ADJ_NODE_ID

The *adj_node_id* (node ID of adjacent node) has already been defined in another link station.

AP_INVALID_LINK_NAME

The link station value specified in the *ls_name* parameter was not valid.

AP_INVALID_NUM_LS_SPECIFIED

The number of link stations specified was not valid.

AP_LOCAL_CP_NAME

The name specified for the *adj_cp_name* parameter is identical to the local CP name.

AP_LS_ACTIVE

The link station specified in the *ls_name* parameter is currently active.

AP_PU_ALREADY_DEFINED

The PU specified in the *pu_name* parameter has already been defined.

AP_DSPU_ALREADY_DEFINED

The downstream PU specified in the *dspu_name* parameter has already been defined.

AP_DSPU_SERVICES_NOT_SUPPORTED

AP_PU_CONCENTRATION or AP_DLUR has been specified on the *dspu_services* parameter, but the node does not support it.

AP_DUPLICATE_TG_NUMBER

The TG number specified in the *tg_number* parameter has already been defined.

AP_TG_NUMBER_IN_USE

The TG number specified for the *tg_number* parameter is already being used by another LS.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Bit Ordering in MAC Addresses

Ethernet LANs use a different representation of MAC addresses from that used by Token Ring and FDDI; the order of the bits in each byte of the address on Ethernet is the reverse of the order on Token Ring. Normally, the local and remote nodes are on the same LAN, or on LANs of the same type connected by a bridge; in this case, they will both use the same representation of the MAC address, and no conversion is required.

If the two nodes are on LANs of different types (one Ethernet, the other Token Ring or FDDI) connected by a bridge, you will normally need to reverse the bit order of each byte of the address when specifying a remote MAC address. To do this, take the following steps:

Step 1. List the MAC address as six bytes, each byte represented by two hexadecimal digits.

Step 2. List the MAC address as six bytes, each byte represented by two

hexadecimal digits.

Step 3. Convert each digit as shown below:

Table 3-1

0 → 0	8 → 1
1 → 8	9 → 9
2 → 4	A → 5
3 → C	B → D
4 → 2	C → 3
5 → A	D → B
6 → 6	E → 7
7 → E	F → F

Table 3-2 Example of Bit Ordering in a MAC Address

Original address	1A	2B	3C	4D	5E	6F
Swap digits	A1	B2	C3	D4	E5	F6
Convert digits (the bit-reversed form of the original address)	58	D4	3C	B2	7A	F6

DEFINE_LU62_TIMEOUT

The DEFINE_LU62_TIMEOUT verb defines a timeout period for unused LU 6.2 sessions. Each timeout is for a specified resource type and resource name. If a DEFINE_* verb is issued for a resource type and name pair already defined, the command overwrites the previous definitions. New timeout periods are only used for sessions activated after the definition is changed.

If more than one relevant timeout period is defined for a session, the shortest period applies.

VCB Structure

```
typedef struct define_lu62_timeout
{
  AP_UINT16         opcode;                    /* verb operation code   */
  unsigned char     reserv2;                   /* reserved              */
  unsigned char     format;                    /* reserved              */
  AP_UINT16         primary_rc;                /* primary return code   */
  AP_UINT32         secondary_rc;              /* secondary return code */
  unsigned char     resource_type;             /* resource type         */
  unsigned char     resource_name[17];         /* resource name         */
  AP_UINT16         timeout;                   /* timeout               */
} DEFINE_LU62_TIMEOUT;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_LU62_TIMEOUT

resource_type Specifies the type of timeout to be defined. Possible values are:

AP_GLOBAL_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local node. The *resource_name* parameter should be set to all zeros.

AP_LOCAL_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local LU specified in the *resource_name* parameter.

AP_PARTNER_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions to the partner LU specified in the *resource_name* parameter.

AP_MODE_TIMEOUT

Timeout applies to all LU 6.2 sessions on the mode specified in the *resource_name* parameter.

resource_name Name of the resource being queried. This value can be one of the following:

- If *resource_type* is set to AP_GLOBAL_TIMEOUT, do not specify this parameter.
- If *resource_type* is set to AP_LOCAL_LU_TIMEOUT, specify 1-8 locally displayable type-A characters as a local LU name.
- If *resource_type* is set to AP_PARTNER_LU_TIMEOUT, specify the fully qualified name of the partner LU as follows: 17 locally displayable type-A characters consisting of a 1-8 character network name, followed by a period, followed by a 1-8 character partner LU name.
- If *resource_type* is set to AP_MODE_TIMEOUT, specify 1-8 locally displayable type-A characters as a mode name.

timeout Timeout period in seconds. A value of 0 (zero) indicates that the session immediately becomes free.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_RESOURCE_TYPE

The type of timeout defined was not valid.

AP_INVALID_LU_NAME

The *resource_type* parameter specified an LU name that was not valid.

AP_INVALID_PARTNER_LU

The *resource_type* parameter specified a partner LU name that was not valid.

AP_INVALID_MODE_NAME

The *resource_type* parameter specified a mode name that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_LU_0_TO_3

The DEFINE_LU_0_TO_3 verb defines an LU for use with 3270 emulation or LUA (an LU of type 0, 1, 2, or 3), and optionally assigns the LU to an LU pool.

If this verb is used to modify an existing LU, only the *description*, *priority*, and *lu_model* parameters can be changed; all other parameters must be set to their existing values.

VCB Structure

```
typedef struct define_lu_0_to_3
{
    AP_UINT16           opcode;                 /* verb operation code      */
    unsigned char       reserv2;                /* reserved                 */
    unsigned char       format;                 /* reserved                 */
    AP_UINT16           primary_rc;             /* primary return code      */
    AP_UINT16           primary_rc;             /* primary return code      */
    AP_UINT32           secondary_rc;           /* secondary return code    */
    unsigned char       lu_name[8];             /* LU name                  */
    LU_0_TO_3_DEF_DATA  def_data;               /* defined data             */
} DEFINE_LU_0_TO_3;

typedef struct lu_0_to_3_def_data
{
    unsigned char       description[32];         /* resource description    */
    unsigned char       reserv1[16];             /* reserved                */
    unsigned char       nau_address;             /* LU NAU address          */
    unsigned char       pool_name[8];            /* LU Pool name            */
    unsigned char       pu_name[8];              /* PU name                 */
    unsigned char       priority;                /* LU priority             */
    unsigned char       lu_model;                /* LU model (type)         */
    unsigned char       sscp_id[6];              /* SSCP ID                 */
    AP_UINT16           timeout;                 /* Timeout                 */
    unsigned char       app_spec_def_data[16];   /* reserved                */
} LU_0_TO_3_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_LU_0_TO_3

lu_name

Name of the local LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the LU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_LU_0_TO_3 verb, but SNAplus2 does not make any other use of it.

def_data.nau_address

Network accessible unit address of the LU. This is a number in the range 1-255.

def_data.pool_name

Name of pool to which this LU belongs. This is an EBCDIC string, padded on the right with EBCDIC spaces if the name is shorter than 8 bytes. If a pool with the specified name is not already defined, SNAplus2 adds a new pool with this name and assigns the LU to it.

If the LU does not belong to a pool, set this field to 8 binary zeros.

def_data.pu_name

Name of the PU (as specified on the DEFINE_LS verb) which this LU will use. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

def_data.priority

LU priority when sending to the host. Possible values are:

AP_NETWORK

AP_HIGH

AP_MEDIUM

AP_LOW

def_data.lu_model

Type of the LU. Possible values are:

AP_3270_DISPLAY_MODEL_2

AP_3270_DISPLAY_MODEL_3

AP_3270_DISPLAY_MODEL_4

AP_3270_DISPLAY_MODEL_5

AP_PRINTER

AP_SCS_PRINTER

AP_RJE_WKSTN

AP_UNKNOWN (LU type will be determined when the session to the host is established)

If the host system supports DDDLU (Dynamic Definition of Dependent LUs), and this parameter is set to any value other than AP_UNKNOWN, SNAplus2 will define the LU dynamically at the host when the communications link to the host is established. If the host does not support DDDLU, or if this parameter is set to AP_UNKNOWN, the LU must be included in the host configuration.

def_data.sscp_id

Specifies the ID of the SSCP permitted to activate this LU. Set this parameter to 0 (zero) if the LU can be activated by any SSCP. If the LU is to be activated only by a specific SSCP, set the first four bytes of this parameter to 0x05000000 and the last two bytes to the SSCP ID that identifies the SSCP that is permitted to activate the LU.

def_data.timeout

Timeout for the LU specified in seconds. If the timeout is set to a nonzero value and the user of the LU specified *allow_timeout* on the

DEFINE_DOWNSTREAM_LU verb, then the LU is deactivated after the PLU-SLU session is left inactive for the specified period and one of the following conditions exist:

- The session passes over a limited resource link.
- Another application requests to use the LU before the session is used again.

If the timeout is set to 0 (zero), the LU is not deactivated.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The *lu_name* parameter contained a character that was not valid.

AP_INVALID_POOL_NAME

The *pool_name* parameter contained a character that was not valid.

AP_INVALID_NAU_ADDRESS

The *nau_address* parameter was not in the permitted range.

AP_INVALID_PRIORITY

The *priority* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *pu_name* parameter was not valid.

AP_PU_NOT_DEFINED

The *pu_name* parameter did not match any defined PU name.

AP_INVALID_PU_TYPE

The PU specified by the *pu_name* parameter is not a host PU.

AP_LU_NAME_POOL_NAME_CLASH

The LU name clashes with the name of an LU pool.

AP_LU_ALREADY_DEFINED

An LU with the specified name has already been defined.

AP_LU_NAU_ADDR_ALREADY_DEFD

An LU with the specified NAU address has already been defined.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_LU_0_TO_3_RANGE

The DEFINE_LU_0_TO_3 verb defines a range of LUs for use with 3270 emulation or LUA (LUs of type 0, 1, 2, or 3), and optionally assigns the LUs to an LU pool. This verb cannot be used to modify existing LUs.

The supplied parameters to this verb include a base name for the new LUs and the range of NAU addresses. The new LU names are generated by combining the base name with the NAU addresses. For example, a base name of LUNME combined with a NAU range of 11 to 14 would define the LUs LUNME011, LUNME012, LUNME013 and LUNME014.

VCB Structure

```
typedef struct define_lu_0_to_3_range
{
     AP_UINT16        opcode;                 /* verb operation code       */
     unsigned char    reserv2;                /* reserved                  */
     unsigned char    format;                 /* reserved                  */
     AP_UINT16        primary_rc;             /* primary return code       */
     AP_UINT32        secondary_rc;           /* secondary return code     */
     unsigned char    base_name[5];           /* Base name                 */
     unsigned char    reserv3;                /* reserved                  */
     unsigned char    description[32];        /* resource description      */
     unsigned char    reserv1[16];            /* reserved                  */
     unsigned char    min_nau;                /* Minimum NAU address       */
     unsigned char    max_nau;                /* Maximum NAU address       */
     unsigned char    pool_name[8];           /* LU Pool name              */
     unsigned char    pu_name[8];             /* PU name                   */
     unsigned char    priority;               /* LU priority               */
     unsigned char    lu_model;               /* LU model (type)           */
     unsigned char    sscp_id[6];             /* SSCP ID                   */
     AP_UINT16        timeout;                /* Timeout                   */
     unsigned char    app_spec_def_data[16];  /* reserved                  */
     unsigned char    name_attributes;        /* Extension type            */
     unsigned char    base_number;            /* First extension number    */
     unsigned char    reserv4[15];            /* reserved                  */
} DEFINE_LU_0_TO_3_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_LU_0_TO_3_RANGE

base_name

Base name for the names of the new LUs. This is a 5-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the base name is less than 5 characters. SNAplus2 generates the LU name for each LU by appending the 3-digit decimal value of the NAU address to this name.

description

A null-terminated text string (0-31 characters followed by a null character) describing the LUs; the same string is used for each LU in the range. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_LU_0_TO_3 verb, but SNAplus2 does not make any other use of it.

min_nau

NAU address of the first LU, in the range 1-255.

max_nau

NAU address of the last LU, in the range 1-255.

pool_name

Name of pool to which these LUs belong. This is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces if the name is shorter than 8 bytes. If a pool with the specified name is not already defined, SNAplus2 adds a new pool with this name and assigns the LUs to it.

If the LUs do not belong to a pool, set this field to 8 binary zeros.

pu_name

Name of the PU (as specified on the DEFINE_LS verb) which these LUs will use. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

priority

LU priority when sending to the host. Possible values are:

AP_NETWORK

AP_HIGH

AP_MEDIUM

AP_LOW

lu_model

Type of the LUs. Possible values are:

AP_3270_DISPLAY_MODEL_2

AP_3270_DISPLAY_MODEL_3

AP_3270_DISPLAY_MODEL_4

AP_3270_DISPLAY_MODEL_5

AP_PRINTER

AP_SSCP_PRINTER

AP_RJE_WKSTN

AP_UNKNOWN (LU type will be determined when the session to the host is established)

If the host system supports DDDLU (Dynamic Definition of Dependent LUs), and this parameter is set to any value other than AP_UNKNOWN, SNAplus2 will define the LUs dynamically at the host when the communications link to the host is established. If the host does not support DDDLU, or if this parameter is set to AP_UNKNOWN, the LUs must be included in the host configuration.

sscp_id

Specifies the ID of the SSCP permitted to activate this LU. Specify a value in the range 0-65,535. If this parameter is set to 0 (zero), the LU can be activated by any SSCP.

timeout

Timeout for the LU specified in seconds. If the timeout is set to a nonzero value and the user of the LU specified *allow_timeout* on the DEFINE_DOWNSTREAM_LU verb, then the LU is deactivated after the PLU-SLU session is left inactive for the specified period and one of the following conditions exist:

- The session passes over a limited resource link.
- Another application requests to use the LU before the session is used again.

If the timeout is set to 0 (zero), the LU is not deactivated.

name_attributes

Attributes of the LUs to be defined. Possible values are:

AP_NONE

LU names have numbers corresponding to the NAU numbers. The numbers are specified in decimal and the *base_name* parameter can only be 5 characters.

AP_USE_BASE_NUMBER

Start naming the LUs in the range from the value specified in the *base_number* parameter.

AP_USE_HEX_IN_NAME

Add the extension to the LU name in hex rather than decimal. The *base_name* parameter can contain 6 characters if this value is specified.

base_number

If AP_USE_BASE_NUMBER is specified in the *name_attributes* parameter, specify a number from which to start naming the LUs in the range. This value will be used instead of the value of the *min_nau* parameter.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The *base_name* parameter contained a character that was not valid.

AP_INVALID_POOL_NAME

The *pool_name* parameter contained a character that was not valid.

AP_INVALID_NAU_ADDRESS

One or more of the LU addresses were not in the permitted range.

AP_INVALID_PRIORITY

The *priority* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *pu_name* parameter was not valid.

AP_PU_NOT_DEFINED

The *pu_name* parameter did not match any defined PU name.

AP_INVALID_PU_TYPE

The PU specified by the *pu_name* parameter is not a host PU.

AP_LU_NAME_POOL_NAME_CLASH

One of the LU names in the range clashes with the name of an LU pool.

AP_LU_ALREADY_DEFINED

An LU has already been defined with the name of one of the LUs in the range.

AP_LU_NAU_ADDR_ALREADY_DEFD

An LU has already been defined with the address of one of the LUs in the range.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_LU_LU_PASSWORD

DEFINE_LU_LU_PASSWORD provides a password which is used for session-level security verification between a local LU and a partner LU.

VCB Structure

```
typedef struct define_lu_lu_password
{
   AP_UINT16        opcode;                  /* verb operation code        */
   unsigned char    reserv2;                 /* reserved                   */
   unsigned char    format;                  /* reserved                   */
   AP_UINT16        primary_rc;              /* primary return code        */
   AP_UINT32        secondary_rc;            /* secondary return code      */
   unsigned char    lu_name[8];              /* local LU name              */
   unsigned char    lu_alias[8];             /* local LU alias             */
   unsigned char    fqplu_name[17];          /* fully qualified partner    */
                                             /*  LU name                   */
   unsigned char    verification_protocol;   /* verfication protocol       */
   unsigned char    description[32];         /* resource description       */
   unsigned char    reserv1[16];             /* reserved                   */
   unsigned char    reserv3[8];              /* reserved                   */
   unsigned char    password[8];             /* password                   */
} DEFINE_LU_LU_PASSWORD;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_LU_LU_PASSWORD

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

verification_protocol

Requested LU-LU verfication protocol to use. Possible values are:

AP_BASIC

Use basic LU-LU verification protocols.

AP_ENHANCED

Use enhanced LU-LU verification protocols.

AP_EITHER

Basic or enhanced verification is accepted.

description

A null-terminated text string (0-31 characters followed by a null character) describing the password. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_LU_LU_PASSWORD verb, but SNAplus2 does not make any other use of it.

password

Password. This is an 8-byte hexadecimal string, which must not be set to all blanks or all zeros. It must match the equivalent parameter configured for the partner LU on the remote system (except that the least

significant bit of each byte is not used in session-level security verification and does not need to match).

Whatever value the application supplies for this parameter is immediately replaced by the encrypted version of the password. Therefore, the value supplied for the *password* parameter is never written out.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined local LU name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_LU_POOL

This verb is used to define an LU pool and assign LUs to it, or to assign additional LUs to an existing pool. The LUs must be defined before adding them to the pool. You can also define a pool by specifying the pool name when defining an LU; For more information, see "DEFINE_LU_0_TO_3".

This verb cannot be used to modify an existing pool by removing LUs from it; the DELETE_LU_POOL verb is used to do this.

VCB Structure

```
typedef struct define_lu_pool
{
     AP_UINT16         opcode;              /* verb operation code          */
     unsigned char     reserv2;             /* reserved                     */
     unsigned char     format;              /* reserved                     */
     AP_UINT16         primary_rc;          /* primary return code          */
     AP_UINT32         secondary_rc;        /* secondary return code        */
     unsigned char     pool_name[8];        /* LU pool name                 */
     unsigned char     description[32];     /* resource description         */
     unsigned char     reserv1[16];         /* reserved                     */
     unsigned char     reserv3[4];          /* reserved                     */
     AP_UINT16         num_lus;             /* number of LUs to add         */
     unsigned char     lu_names[10][8];     /* LU names                     */
} DEFINE_LU_POOL;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_LU_POOL

pool_name Name of the LU pool. This is an 8-byte type-AE EBCDIC string, padded on the right with EBCDIC spaces if the name is shorter than 8 bytes. If a pool of this name is not already defined, SNAplus2 creates it.

description A null-terminated text string (0-31 characters followed by a null character) describing the pool. This string is for information only; it is stored in the node's

configuration file and returned on the QUERY_LU_POOL verb, but SNAplus2 does not make any other use of it.

num_lus Number of LUs to be added to the pool. This can be zero to define the pool without adding any LUs, or 1-10. To create a pool containing more than 10 LUs, issue multiple DEFINE_LU_POOL verbs for the same pool name.

lu_names Names of the LUs that are being assigned to the pool. Each of these LUs must already be defined to SNAplus2 as an LU of type 0-3. Each LU name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

If a specified LU is currently assigned to a different pool, SNAplus2 removes it from that pool (because an LU cannot be in more than one pool) and assigns it to the pool specified by this verb.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

One or more of the supplied LU names did not match any defined LU name.

AP_INVALID_POOL_NAME

The *pool_name* parameter contained a character that was not valid.

AP_INVALID_NUM_LUS

The *num_lus* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_LU_NAME_POOL_NAME_CLASH

The specified pool name clashes with the name of an LU.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_MODE

The DEFINE_MODE verb defines a mode (a set of networking characteristics to be used by a group of sessions) or modifies a previously defined mode. You cannot modify the SNA-defined mode CPSVCMG or change the COS name used by the SNA-defined mode SNASVCMG.

If you use this verb to modify an existing mode, the changes will apply to any new combination of local LU and partner LU that start to use the mode after you have made the change. However, any combination of LUs already using the mode will not pick up the changes until after the next locally or remotely initiated CNOS command.

This verb can also be used to specify the default COS to which any unrecognized modes will be mapped. If no default COS is specified, the SNA-defined COS #CONNECT is used.

VCB Structure

```
typedef struct define_mode
{
  AP_UINT16       opcode;               /* verb operation code            */
  unsigned char   reserv2;              /* reserved                       */
  unsigned char   format;               /* reserved                       */
  AP_UINT16       primary_rc;           /* primary return code            */
  AP_UINT32       secondary_rc;         /* secondary return code          */
  unsigned char   mode_name[8];         /* mode name                      */
  AP_UINT16       reserv3;              /* reserved                       */
  MODE_CHARS      mode_chars;           /* mode characteristics           */
} DEFINE_MODE;

typedef struct mode_chars
{
  unsigned char   description;          /* resource description           */
  unsigned char   reserv2[16];          /* reserved                       */
  AP_UINT16       max_ru_size_upp;      /* maximum RU size upper bound    */
  unsigned char   receive_pacing_win;   /* receive pacing window          */
  unsigned char   default_ru_size;      /* default RU size to maximize    */
                                        /*  performance                   */
  AP_UINT16       max_neg_sess_lim;     /* maximum negotiable session limit*/
  AP_UINT16       plu_mode_session_limit; /* LU-mode session limit        */
  AP_UINT16       min_conwin_src;       /* minimum source contention winner*/
                                        /*  sessions                      */
```

```
  unsigned char   cos_name[8];            /* class of service name          */
  unsigned char   cryptography;           /* reserved                       */
  unsigned char   compression;            /* reserved                       */
  unsigned char   reserv1;                /* reserved                       */
  AP_UINT16       auto_act;               /* initial auto-activation count  */
  AP_UINT16       min_conloser_src;       /* min source contention loser    */
  AP_UINT16       max_ru_size_low;        /* maximum RU size lower bound    */
  AP_UINT16       max_receive_pacing_win; /* maximum receive pacing window  */
} MODE_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_MODE

mode_name

Name of the mode. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

To specify the default COS that will be used for any unrecognized mode names, set this parameter to 8 binary zeros. In this case, the *mode_chars.cos_name* parameter is taken as the default COS name; all other parameters supplied on this verb are ignored.

mode_chars.description

A null-terminated text string (0-31 characters followed by a null character) describing the mode. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_MODE_DEFINITION and QUERY_MODE verbs, but SNAplus2 does not make any other use of it.

mode_chars.max_ru_size_upp

Upper bound for the maximum size of RUs sent and received on sessions in this mode. The value is used when the maximum RU size is negotiated during session activation.

Range: 256-61,440. If the *default_ru_size*

parameter (see below) is set to AP_YES, this parameter is ignored (and the value is not checked).

mode_chars.receive_pacing_win

Session pacing window for sessions using this mode; the range is 1-63. This value is used only for fixed pacing (not for adaptive pacing), and specifies the maximum number of frames that can be received from the partner LU before the local LU must send a response. SNAplus2 always uses adaptive pacing unless the adjacent node specifies that it is not supported.

mode_chars.default_ru_size

Specifies whether a default upper bound for the maximum RU size will be used. Possible values are:

AP_YES

SNAplus2 ignores the *max_ru_size_upp* parameter, and sets the upper bound for the maximum RU size to the largest value that can be accommodated in the link BTU size.

AP_NO

SNAplus2 uses the *max_ru_size_upp* parameter to define the maximum RU size.

mode_chars.max_neg_sess_lim

Maximum number of sessions allowed on this mode between any local LU and partner LU. This value may be lowered for a particular LU-LU-mode combination when issuing *initialize_session_limit* or *change_session_limit*.

Range: 1-32,767. Zero indicates that SNAplus2 should not initiate implicit CNOS exchange when an application attempts to start a session using this mode; session limits must be specified explicitly using *initialize_session_limit*.

mode_chars.plu_mode_session_limit

Default session limit for this mode. This limits the number of sessions on this mode between any one local

LU and partner LU pair. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly.

Specify a value in the range 1-32,767 (which must not exceed the value in *max_neg_sess_lim*

above). Zero indicates that SNAplus2 should not initiate implicit CNOS exchange when an application attempts to start a session using this mode; session limits must be specified explicitly using *initialize_session_limit*.

If you specify an explicit limit, the LU session limit for any LU that uses this mode must be greater than or equal to the sum of the session limits for all modes that the LU will use.

mode_chars.min_conwin_src

Minimum number of contention winner sessions that a local LU using this mode can activate. This value is used when CNOS (Change Number of Sessions) exchange is initiated either by the remote system or implicitly by SNAplus2. Specify a value in the range 0-32,767.

mode_chars.cos_name

Name of the class of service to request when activating sessions on this mode.

If the node supports mode to COS mapping (as defined by the *mode_to_cos_map_supp* parameter on DEFINE_NODE), the COS specified by this field must be either an SNA defined COS or a COS previously defined by issuing a DEFINE_COS verb. Otherwise, this parameter is ignored.

The name is an 8-byte type-A character string, padded on the right with spaces if the name is shorter than 8 characters.

mode_chars.auto_act

Number of sessions that will be activated automatically for this mode. This value is used when CNOS (Change Number of Sessions) exchange is

initiated implicitly. Specify a value in the range 0-32,767.

mode_chars.min_conloser_src

Minimum number of contention loser sessions that can be activated by any one local LU that uses this mode. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. Specify a value in the range 0-32,767.

mode_chars.max_ru_size_low

Lower bound for the maximum size of RUs sent and received on sessions that use this mode. Specify a value in the range 256-61,440. The value 0 means that there is no lower bound.

The value is used when the maximum RU size is negotiated during session activation. This parameter is ignored if the *default_ru_size* parameter is set to AP_YES.

mode_chars.max_receive_pacing_win

Maximum session pacing window for sessions in this mode. For adaptive pacing, this value is used to limit the receive pacing window that the session will grant. For fixed pacing, this parameter is not used. (SNAplus2 always uses adaptive pacing unless the adjacent node specifies that it does not support it.)

Specify a value in the range 0-32,767. The value zero means that there is no upper bound.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_CPSVCMG_ALREADY_DEFD

The SNA-defined mode CPSVCMG cannot be changed.

AP_INVALID_CNOS_SLIM

The *plu_mode_session_limit* parameter is not valid.

AP_INVALID_COS_NAME

The *cos_name* parameter did not match any defined COS name.

AP_INVALID_COS_SNASVCMG_MODE

The COS for the SNA-defined mode SNASVCMG cannot be changed.

AP_INVALID_DEFAULT_RU_SIZE

The *default_ru_size* parameter was not in the valid range.

AP_INVALID_MAX_NEGOT_SESS_LIM

The *max_neg_sess_lim* parameter was not in the valid range.

AP_INVALID_MAX_RU_SIZE_UPPER

The *max_ru_size_upp* parameter was not in the valid range.

AP_INVALID_MIN_CONWINNERS

The *min_conwin_src* parameter was not in the valid range.

AP_INVALID_MODE_NAME

The *mode_name* parameter contained a character that was not valid.

AP_INVALID_RECV_PACING_WINDOW

The *receive_pacing_win* parameter was not in the valid range.

AP_INVALID_SNASVCMG_MODE_LIMIT

The SNA-defined mode SNASVCMG must have a session limit of 2 and *min_conwin_src* of 1. You cannot define SNASVCMG with different values for these parameters.

AP_MODE_SESS_LIM_EXCEEDS_NEG

The value specified for *plu_mode_session_limit* was larger than the value specified for *max_neg_sess_lim*.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_NODE

An application issues this verb in order to define a new node, or to modify the parameters of an inactive node.

This verb must be issued to a server where the node is not running. It cannot be issued to a running node.

VCB Structure

```
typedef struct define_node
{
  AP_UINT16         opcode;             /* verb operation code              */
  unsigned char     reserv2;            /* reserved                         */
  unsigned char     format;             /* reserved                         */
  AP_UINT16         primary_rc;         /* primary return code              */
  AP_UINT32         secondary_rc;       /* secondary return code            */
  unsigned char     node_name[64];      /* name of Node                     */
  AP_UINT32         target_handle;      /* handle for subsequent verbs      */
  CP_CREATE_PARMS   cp_create_parms;    /* CP create parameters             */
} DEFINE_NODE;

typedef struct cp_create_parms
{
  AP_UINT16       crt_parms_len;        /* length of CP_CREATE_PARMS        */
  unsigned char   description[32];       /* resource description            */
  unsigned char   reserv1[2];           /* reserved                         */
  unsigned char   ms_support;           /* reserved                         */
  unsigned char   queue_nmvts;          /* reserved                         */
  unsigned char   reserv3[12];          /* reserved                         */
  unsigned char   node_type;            /* node type                        */
  unsigned char   fqcp_name[17];        /* fully qualified CP name          */
  unsigned char   cp_alias[8];          /* CP alias                         */
  unsigned char   mode_to_cos_map_supp; /* mode to COS mapping support      */
  unsigned char   mds_supported;        /* MDS and MS capabilities          */
  unsigned char   node_id[4];           /* node ID                          */
  AP_UINT16       max_locates;          /* maximum locates node can process */
  AP_UINT16       dir_cache_size;       /* directory cache size (reserved   */
                                        /*  is not NN)                      */
  AP_UINT16       max_dir_entries;      /* maximum directory entries (zero  */
                                        /*  means unlimited)                */
  AP_UINT16       locate_timeout;       /* locate timeout in seconds        */
  unsigned char   reg_with_nn;          /* register resources with NNS      */
```

```
  unsigned char   reg_with_cds;         /* register resources with CDS       */
  AP_UINT16       mds_send_alert_q_size;/* size of MDS send alert queue      */
  AP_UINT16       cos_cache_size;       /* number of cos definitions         */
  AP_UINT16       tree_cache_size;      /* reserved                          */
  AP_UINT16       tree_cache_use_limit; /* reserved                          */
  AP_UINT16       max_tdm_nodes;        /* reserved                          */
  AP_UINT16       max_tdm_tgs;          /* reserved                          */
  AP_UINT32       max_isr_sessions;     /* reserved                          */
  AP_UINT32       isr_sessions_upper_threshold;  /* reserved                 */
  AP_UINT32       isr_sessions_lower_threshold;  /* reserved                 */
  AP_UINT16       isr_max_ru_size;      /* reserved                          */
  AP_UINT16       isr_rcv_pac_window;   /* reserved                          */
  unsigned char   store_endpt_rscvs;    /* endpoint RSCV storage             */
  unsigned char   store_isr_rscvs;      /* reserved                          */
  unsigned char   store_dlur_rscvs;     /* DLUR RSCV storage                 */
  unsigned char   dlur_support;         /* is DLUR supported?                */
  unsigned char   pu_conc_support;      /* is PU conc supported?             */
  unsigned char   nn_rar;               /* Route additional resistance       */
  unsigned char   hpr_support;          /* reserved                          */
  unsigned char   mobile;               /* reserved                          */
  unsigned char   discovery_support;    /* reserved                          */
  unsigned char   discovery_group_name[8]; /* reserved                       */
  unsigned char   implicit_lu_0_to_3;   /* reserved                          */
  unsigned char   default_preference;   /* reserved                          */
  unsigned char   anynet_supported;     /* reserved                          */
#define MAX_LS_EXCEPTION_EVENTS 200
  AP_UINT16       max_ls_exception_events  /* Max # exception entries         */
  unsigned char   reserv2[1];              /* reserved                        */
  unsigned char   max_compress_lvl;        /* reserved                        */
  unsigned char   node_spec_data_len;      /* reserved                        */
  unsigned char   ptf[64];                 /* program temporary fix array     */
} CP_CREATE_PARMS;
```

Supplied Parameters

opcode

AP_DEFINE_NODE

node_name

Name of SNAplus2 node that the application wishes to define.

cp_create_parms.crt_parms_len

Length of create parameters structure.

cp_create_parms.description

A text string (0-31 characters followed by a null character) describing the node. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_NODE verb, but SNAplus2 does not make any other use of it.

cp_create_parms.node_type

One of the following node types:

AP_END_NODE

AP_LEN_NODE

cp_create_parms.fqcp_name

Node's fully qualified CP name. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

cp_create_parms.cp_alias

Locally used CP alias. This alias can be used by APPC applications to access the CP LU. This is an 8-byte ASCII string. All 8 bytes are significant and must be set.

cp_create_parms.mode_to_cos_map_supp

Specifies whether the node provides mode-to-COS mapping. For a LEN node, mode-to-COS mapping is not supported. Possible values are:

AP_YES

Mode-to-COS mapping is supported. A mode defined for this node must include an associated COS name, which specifies either an SNA-defined COS or one defined using DEFINE_COS.

AP_NO

Mode-to-COS mapping is not supported. Default COS names will be used.

cp_create_parms.mds_supported

Specifies whether Management Services supports Multiple Domain Support and MS Capabilities. Possible values are:

AP_YES

MDS is supported.

AP_NO

MDS is not supported.

cp_create_parms.node_id

Node identifier used in XID exchange. This is a 4-byte hexadecimal string, consisting of a block number (3 hexadecimal digits) and a node number (5 hexadecimal digits).

cp_create_parms.max_locates

Maximum number of locate requests that the node can process simultaneously. When the number of outstanding locate requests (requests for which a response has not yet been received) reaches this limit, any further locate requests are rejected. The minimum is 8.

cp_create_parms.dir_cache_size

Reserved (set this parameter to zero).

cp_create_parms.max_dir_entries

Maximum number of directory entries. Specify zero for no limit.

cp_create_parms.locate_timeout

Specifies the time in seconds before a network search will timeout. Specify zero for no timeout.

cp_create_parms.reg_with_nn

End node only: Specifies whether to register the node's resources with the network node server when the node is started. Possible values are:

AP_YES

Register resources with the NN. The end node's network node server will only forward directed locates to it.

AP_NO

Do not register resources. The network node server will forward all broadcast searches to the end node.

cp_create_parms.reg_with_cds

End node: Specifies whether the network node server is allowed to register end node resources with a Central Directory server. This field is ignored if *reg_with_nn* is set to AP_NO.

Possible values are:

AP_YES

Register resources with the CDS.

AP_NO

Do not register resources.

cp_create_parms.mds_send_alert_q_size

Size of the MDS send alert queue. If the number of queued alerts reaches this limit, SNAplus2 deletes the oldest alert on the queue. SNAplus2 uses the value 2 unless you specify a larger number.

cp_create_parms.cos_cache_size

Size of the COS Database weights cache. This value should be set to the maximum number of COS definitions required. SNAplus2 uses the value 8 unless you specify a larger number.

cp_create_parms.store_endpt_rscvs

Specifies whether RSCVs should be stored for diagnostic purposes (AP_YES or AP_NO). If this field is set to AP_YES, then an RSCV will be returned on the QUERY_SESSION verb. (Setting this value to AP_YES means an RSCV will be stored for each endpoint session. This extra storage can be up to 256 bytes per session.)

cp_create_parms.store_dlur_rscvs

Specifies whether RSCVs should be stored for diagnostic purposes (AP_YES or AP_NO). If this field is set to AP_YES, then an RSCV will be returned on the QUERY_DLUR_LU verb. (Setting this value to AP_YES means an RSCV will be stored for each PLU-SLU session using DLUR. This extra storage can be up to 256 bytes per session.)

cp_create_parms.dlur_support

Specifies whether DLUR is supported (AP_YES or AP_NO). For a LEN node, this parameter is reserved.

cp_create_pams.pu_conc_support

Specifies whether PU concentration is supported (AP_YES or AP_NO).

cp_create_parms.nn_rar

Reserved (set this parameter to zero).

cp_create_parms.hpr_support

This parameter is reserved

cp_create_parms.max_ls_exception_events

The maximum number of LS exception events to be recorded by the node.

cp_create_parms.ptf

Array for configuring and controlling future program temporary fix (ptf) operation, as follows:

cp_create_parms.ptf[0]

REQDISCONT support. SNAplus2 normally uses REQDISCONT to deactivate limited resource host links that are no longer required by session traffic. This byte can be used to suppress use of REQDISCONT, or to modify the settings used on REQDISCONT requests sent by SNAplus2. Set this byte to one of the following values:

AP_NONE

Use the normal REQDISCONT support.

AP_SUPPRESS_REQDISCONT

Do not use REQDISCONT.

AP_OVERRIDE_REQDISCONT

Use a modified version of REQDISCONT support. If REQDISCONT is specified, it must be combined with one or both of the following values, using a logical OR operation:

AP_REQDISCONT_TYPE

Use type “immediate” on REQDISCONT; if this value is not specified, SNAplus2 uses type “normal”.

AP_REQDISCONT_RECONTACT

Use type “immediate recontact” on REQDISCONT; if this value is not specified, SNAplus2 uses type “no immediate recontact”.

AP_ALLOW_BB_RQE

SNAplus2 normally rejects, with sense code 2003, any begin bracket (BB) exception (RQE) request from a host unless the host follows the SNA protocol that the request must also indicate change direction (CD). Setting this flag enables SNAplus2 to continue sessions with hosts that do not follow this protocol.

cp_create_parms.ptf[1]

ERP support. SNAplus2 normally processes an ACTPU(ERP) as an ERP; this resets the PU-SSCP session, but does not implicitly deactivate the subservient LU-SSCP and PLU-SLU sessions. SNA implementations may legally process ACTPU(ERP) as if it were ACTPU(cold), implicitly deactivating the subservient LU-SSCP and PLU-SLU sessions. Set this byte to one of the following values:

AP_NONE

Use the normal processing.

AP_OVERRIDE_ERP

Process all ACTPU requests as ACTPU(cold).

cp_create_parms.ptf[2]

BIS support. SNAplus2 normally uses the BIS protocol prior to deactivating a limited resource LU 6.2 session. Set this byte to one of the following values:

AP_NONE

Use the normal processing.

AP_SUPPRESS_BIS

Do not use the BIS protocol. Limited resource LU 6.2 sessions are deactivated immediately using UNBIND(cleanup).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

target_handle Returned value for use on subsequent verbs.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_ISR_THRESHOLDS

The ISR threshold parameters were not valid (lower threshold above upper, or upper threshold above *max_isr_sessions*).

AP_INVALID_NODE_NAME

The *node_name* parameter contained a character that was not valid.

AP_INVALID_CP_NAME

The *cp_alias* or *fqcp_name* parameter contained a character that was not valid.

AP_INVALID_NODE_TYPE

The *node_type* parameter was not set to a valid value.

AP_PU_CONC_NOT_SUPPORTED

This version of SNAplus2 does not support the PU concentration feature.

AP_DLUR_NOT_SUPPORTED

This version of SNAplus2 does not support the DLUR feature.

AP_HPR_NOT_SUPPORTED

You specified a nonzero value for a reserved parameter.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_NODE_ALREADY_STARTED

The target node is active, so you cannot use this verb to modify its configuration. DEFINE_NODE can be issued only to an inactive node.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_PARTNER_LU

The DEFINE_PARTNER_LU verb defines the parameters of a partner LU for LU-LU sessions between a local LU and the partner LU, or modifies an existing partner LU. You cannot change the partner LU alias of an existing partner LU.

There is normally no requirement to define partner LUs, because SNAplus2 will set up an implicit definition when the session to the partner LU is established; you should only need to define the LU if you need to enforce non-default values for logical record size, conversation security support, or parallel session support. You may also have an APPC application that uses a partner LU alias when allocating a session, therefore you need to define a partner LU in order to map the alias to a fully-qualified partner LU name.

If the local node or the remote node (where the partner LU is located) is a LEN node, note that you need to define a directory entry for the partner LU to allow SNAplus2 to access it. This can be done using either DEFINE_ADJACENT_LEN_NODE or DEFINE_DIRECTORY_ENTRY. If each of the two nodes is either an end node or network node, the directory entry is not required, because SNAplus2 can locate the LU dynamically.

VCB Structure

```
typedef struct define_partner_lu
{
   AP_UINT16        opcode;              /* verb operation code            */
   unsigned char    reserv2;             /* reserved                       */
   unsigned char    format;              /* reserved                       */
   AP_UINT16        primary_rc;          /* primary return code            */
   AP_UINT32        secondary_rc;        /* secondary return code          */
   PLU_CHARS        plu_chars;           /* partner LU characteristics     */
} DEFINE_PARTNER_LU;


typedef struct plu_chars
{
   unsigned char    fqplu_name[17];      /* fully qualified partner LU name */
   unsigned char    plu_alias[8];        /* partner LU alias               */
   unsigned char    description[32];     /* resource description           */
   unsigned char    reserv2[16];         /* reserved                       */
```

```
   unsigned char    plu_un_name[8];       /* partner LU uninterpreted name   */
   unsigned char    appcip_routing_preference; /* reserved                   */
   AP_UINT16        max_mc_ll_send_size;  /* maximum MC send LL size         */
   unsigned char    conv_security_ver;    /* already-verified security       */
                                          /*  supported?                     */
   unsigned char    parallel_sess_supp;   /* parallel sessions supported?    */
   unsigned char    reserv3[8];           /* reserved                        */
} PLU_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_PARTNER_LU

plu_chars.fqplu_name

Fully qualified LU name for the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

plu_chars.plu_alias

LU alias of the partner LU. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes.

If the *fqplu_name* parameter above matches the fully qualified name of an existing partner LU, this parameter must match the partner LU alias in the existing definition. You cannot change the partner LU alias for an existing partner LU, or set up more than one LU alias for the same fully qualified name.

plu_chars.description

A null-terminated text string (0-31 characters followed by a null character) describing the partner LU. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_PARTNER_LU and

QUERY_PARTNER_LU_DEFINITION verbs, but SNAplus2 does not make any other use of it.

plu_chars.plu_un_name

Uninterpreted name of the partner LU (the name of the LU as defined to the remote SSCP). The name is an 8-byte EBCDIC character string.

To use the default uninterpreted name (the same as the network name taken from the *fqplu_name* parameter above), set this parameter to 8 binary zeros. This parameter is only relevant if the partner LU is on a host and dependent LU 6.2 is used to access it.

plu_chars.max_mc_ll_send_size

The maximum size of logical records that can be sent and received by mapped conversation services at the partner LU. Specify a number in the range 1-32,767, or zero to specify no limit (in this case the maximum is 32,767).

plu_chars.conv_security_ver

Specifies whether the partner LU is authorized to validate user IDs on behalf of local LUs; that is, whether the partner LU may set the already verified indicator in an Attach request. Possible values are:

AP_YES

The partner LU can validate user IDs.

AP_NO

The partner LU cannot validate user IDs.

plu_chars.parallel_sess_supp

Specifies whether the partner LU supports parallel sessions. Possible values are:

AP_YES

The partner LU supports parallel sessions.

AP_NO

The partner LU does not support parallel sessions.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_DEF_PLU_INVALID_FQ_NAME

The *fqplu_name* parameter contained a character that was not valid.

AP_INVALID_UNINT_PLU_NAME

The *plu_un_name* parameter contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_PLU_ALIAS_CANT_BE_CHANGED

The *plu_alias* parameter of an existing partner LU cannot be changed.

AP_PLU_ALIAS_ALREADY_USED

The *plu_alias* parameter is already used for an existing partner LU with a different LU name.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_PORT

DEFINE_PORT is used to define a new port or modify an existing one. Before issuing this verb, you must issue the DEFINE_DLC verb to define the DLC that this port uses.

You can modify an existing port only if it is not started. You cannot change the DLC used by an existing port; the *dlc_name* specified when modifying an existing port must match the DLC that was specified on the initial definition of the port.

If you are defining a port that will accept incoming calls, see “Incoming Calls”.

VCB Structure

```
typedef struct define_port
{
  AP_UINT16       opcode;                 /* verb operation code            */
  unsigned char   reserv2;                /* reserved                       */
  unsigned char   format;                 /* reserved                       */
  AP_UINT16       primary_rc;             /* primary return code            */
  AP_UINT32       secondary_rc;           /* secondary return code          */
  unsigned char   port_name[8];           /* name of port                   */
  PORT_DEF_DATA   def_data;               /* port defined data              */
} DEFINE_PORT;


typedef struct port_def_data
  unsigned char   description[32];         /* resource description          */
  unsigned char   initially_active;        /* is the port initially active?  */
  unsigned char   reserv2[15];             /* reserved                       */
  unsigned char   dlc_name[8];             /* DLC name associated with port  */
  unsigned char   port_type;               /* port type                      */
  unsigned char   port_attributes[4];      /* port attributes                */
  unsigned char   reserv3[3];              /* reserved                       */
  AP_UINT32       port_number;             /* port number                    */
  AP_UINT16       max_rcv_btu_size;        /* max receive BTU size           */
  AP_UINT16       tot_link_act_lim;        /* total link activation limit    */
  AP_UINT16       inb_link_act_lim;        /* inbound link activation limit  */
  AP_UINT16       out_link_act_lim;        /* outbound link activation limit */
  unsigned char   ls_role;                 /* initial link station role      */
  unsigned char   reserv1[15];             /* reserved                       */
  unsigned char   implicit_dspu_template[8]; /*implicit_dspu_template       */
```

```
  AP_UINT16  implicit_ls_limit;       /* implicit ls limit              */
  unsigned char   reserv4;                /* reserved                       */
  unsigned char   implicit_dspu_services;  /* implicit DSPU support         */
  AP_UINT16  implicit_deact_timer;    /* deact timer for implicit LSs   */
  AP_UINT16       act_xid_exchange_limit;  /* activation XID exchange limit  */
  AP_UINT16       nonact_xid_exchange_limit;     /* non-activation XID       */
                                                 /*  exchange limit          */
  unsigned char   ls_xmit_rcv_cap;        /* LS transmit-receive capability */
  unsigned char   max_ifrm_rcvd;          /* maximum number of I-frames that*/
                                          /*  can be received               */
  AP_UINT16       target_pacing_count;    /* Target pacing count            */
  AP_UINT16       max_send_btu_size;      /* Desired maximum send BTU size  */
  LINK_ADDRESS    dlc_data;               /* DLC data                       */
  LINK_ADDRESS    hpr_dlc_data;           /* reserved                       */
  unsigned char   implicit_cp_cp_sess_support;   /* implicit links allow     */
                                                 /*  CP-CP sessions          */
  unsigned char   implicit_limited_resource;     /* implicit links are       */
                                                 /*  limited resource        */
  unsigned char   implicit_hpr_support;   /* reserved                       */
  unsigned char   implicit_link_lvl_error; /* reserved                      */
  unsigned char   retired1;               /*  reserved                      */
  TG_DEFINED_CHARS default_tg_chars;      /* default TG chars               */
  unsigned char   discovery_supported;    /* reserved                       */
  AP_UINT16       port_spec_data_len;     /* length of port specification   */
                                          /*  data                          */
  AP_UINT16       link_spec_data_len;     /* length of link specification   */
                                          /*  data                          */
} PORT_DEF_DATA;


typedef struct link_address
{
  AP_UINT16       reserve1;               /* reserved                       */
  AP_UINT16       length;                 /* length                         */
  unsigned char   address[32];            /* address                        */
} LINK_ADDRESS;


typedef struct tg_defined_chars
{
  unsigned char   effect_cap;             /* effective capacity             */
  unsigned char   reserve1[5];            /* reserved                       */
  unsigned char   connect_cost;           /* connection cost                */
  unsigned char   byte_cost;              /* byte cost                      */
  unsigned char   reserve2;               /* reserved                       */
  unsigned char   security;               /* security                       */
  unsigned char   prop_delay;             /* propagation delay              */
```

```
  unsigned char   modem_class;            /* reserved                      */
  unsigned char   user_def_parm_1;        /* user-defined parameter 1      */
  unsigned char   user_def_parm_2;        /* user-defined parameter 2      */
  unsigned char   user_def_parm_3;        /* user-defined parameter 3      */
} TG_DEFINED_CHARS;
```

Port-specific data for SDLC:

```
typedef struct sdl_port_spec_data
{
  V0_MUX_INFO     mux_info;           /* Streams config info               */
  AP_UINT32       idle_timer;         /* idle timer (in ms)                */
  AP_UINT16       idle_timer_retry;   /* idle timer retry                  */
  AP_UINT16       reserve1;           /* reserved                          */
  AP_UINT32       np_rcv_timer;       /* non-productive receive timer( in ms)*/
  AP_UINT16       np_rcv_timer_retry; /* non-productive receive timer retry  */
  AP_UINT16       reserve2;           /* reserved                          */
  AP_UINT32       write_timer;        /* write timer (in ms)               */
  AP_UINT16       write_timer_retry;  /* write timer retry                 */
  AP_UINT16       reserve3;           /* reserved                          */
  AP_UINT32       link_conn_timer;    /* link connection timer (in ms)     */
  AP_UINT16       link_conn_timer_retry;  /* link connection timer retry   */
  AP_UINT16       reserve4;           /* reserved                          */
  AP_UINT16       pri_fdplx;          /* Is primary on this link full-duplex */
  AP_UINT16       sec_fdplx;          /* Is secondary on link full-duplex  */
  AP_UINT16       use_rej;            /* Can REJ command be used on this port*/
  AP_UINT16       port_type;          /* Port type                         */
  AP_UINT16       max_xid_size;       /* max size of XIDs in MU_SEND_XID   */
  AP_UINT16       max_retry_count;    /* max number of times to retransmit */
  AP_UINT16       physical_link;      /* reserved                          */
  AP_UINT16       stub_spec_data_len; /* length of next field              */
  STUB_SPEC_DATA  stub_spec_data;     /* data specific to HMOD stub        */
} SDL_PORT_SPEC_DATA;
```

Link-specific data for SDLC:

```
typedef struct sdl_link_spec_data
{
  V0_MUX_INFO     mux_info;             /* Streams config info           */
  AP_UINT16       reserve8;             /* reserved                      */
  AP_UINT16       reserve9;             /* reserved                      */
  AP_UINT32       contact_timer;        /* contact timer (fast poll, in ms)*/
  AP_UINT16       contact_timer_retry;  /* contact timer retry           */
  AP_UINT16       reserve1;             /* reserved                      */
  AP_UINT32       contact_timer2;       /* contact timer (slow poll, in ms)*/
  AP_UINT16       contact_timer_retry2; /* contact timer 2 retry         */
```

```
  AP_UINT16       reserve2;              /* reserved                          */
  AP_UINT32       disc_timer;            /* disconnect timer (in ms)          */
  AP_UINT16       disc_timer_retry;      /* disconnect timer retry            */
  AP_UINT16       reserve3;              /* reserved                          */
  AP_UINT32       nve_poll_timer;        /* negative poll timer (fast poll)   */
  AP_UINT16       nve_poll_timer_retry;  /* negative poll timer retry         */
  AP_UINT16       reserve4;              /* reserved                          */
  AP_UINT32       nve_poll_timer2;       /* negative poll timer (slow poll)   */
  AP_UINT16       nve_poll_timer_retry2; /* negative poll timer 2 retry       */
  AP_UINT16       reserve5;              /* reserved                          */
  AP_UINT32       no_resp_timer;         /* No response timer (T1 timer)      */
                                         /*  (in ms)                          */
  AP_UINT16       no_resp_timer_retry;   /* No response timer retry           */
  AP_UINT16       reserve6;              /* reserved                          */
  AP_UINT32       rem_busy_timer;        /* Remote busy timer (in ms)         */
  AP_UINT16       rem_busy_timer_retry;  /* Remote busy timer retry           */
  unsigned char   re_tx_threshold;       /* I-frame retransmission threshold  */
  unsigned char   repoll_threshold;      /* Poll retransmission threshold     */
  AP_UINT32       rr_timer;              /* RR turnaround timer (in ms)       */
  unsigned char   group_address;         /* reserved                          */
  unsigned char   poll_frame;            /* Poll frame to use when Primary    */
                                         /*  and polling secondary            */
                                         /*  XID, DISC, SNRM, SNRME, TEST     */
  AP_UINT16       poll_on_iframe;        /* Can LS send poll bit on I-frame   */
  AP_UINT16       stub_spec_data_len;    /* length of next field              */
  STUB_SPEC_DATA  stub_spec_data;        /* data specific to HMOD stub        */
} SDL_LINK_SPEC_DATA;

typedef struct stub_spec_data
{
  int             mux_id;          /* reserved                                */
  unsigned char   opt1;            /* options flag 1                          */
  unsigned char   opt2;            /* options flag 2                          */
  unsigned char   pad[2];          /* reserved                                */
  AP_UINT32       linesp;          /* line speed in bps                       */
  AP_UINT16       rcv_pool_size;   /* initial number of buffers for rcv pool  */
  AP_UINT16       poll_wait;       /* seconds between polling HMOD for errors */
  AP_UINT16       hmod_data_len;   /* length of dial data string              */
  unsigned char   hmod_data[80];   /* dial data string                        */
  unsigned char   reserve1[3];     /* reserved                                */
} STUB_SPEC_DATA;
```

Port-specific data for QLLC:

```
typedef struct vql_port_spec_data
```

```
{
  V0_MUX_INFO     mux_info;        /* streams config info                    */
  unsigned char   driver_name[13]; /* reserved                               */
  unsigned char   cud_mode;        /* matching required on CUD               */
  AP_UINT16       cud_len;         /* length of Call User Data in octets     */
  unsigned char   cud[16];         /* Call User Data                         */
  unsigned char   add_mode;        /* matching reqd on called address        */
  AP_UINT16       add_len;         /* length of called address to match      */
  AP_UINT32       xtras;           /* reserved                               */
  AP_UINT32       xtra_len;        /* reserved                               */
} VQL_PORT_SPEC_DATA;
```

Link-specific data for QLLC:

```
typedef struct vql_ls_spec_data
{
  V0_MUX_INFO     mux_info;        /* streams config info                    */
  AP_UINT16       reserve1;        /* reserved                               */
  AP_UINT16       reserve2;        /* reserved                               */
  unsigned char   vc_type;         /* Virtual Circuit type                   */
  unsigned char   req_rev_charge;  /* request reverse charge if non-zero     */
  unsigned char   loc_packet;      /* loc->rem packet size 2**locpacket      */
  unsigned char   rem_packet;      /* rem->loc packet size 2**rempacket      */
  unsigned char   loc_wsize;       /* loc->rem window size                   */
  unsigned char   rem_wsize;       /* rem->loc window size                   */
  AP_UINT16       fac_len;         /* X.25 facilities length                 */
  unsigned char   fac[32];         /* X.25 facilities                        */
  AP_UINT16       retry_limit;     /* times to retry send QXID,QSM,QDISC     */
  AP_UINT16       retry_timeout;   /* timeout for each of above tries        */
  AP_UINT16       idle_timeout;    /* timeout for no Q msgs during init      */
  AP_UINT16       pvc_id;          /* PVC logical channel identifier         */
  AP_UINT16       sn_id_len;       /* Length of the subnet identifier        */
  unsigned char   sn_id[4];        /* Subnet identifier                      */
  AP_UINT16       cud_len;         /* length of any call user data to send   */
  unsigned char   cud[16];         /* actual call user data                  */
  AP_UINT32       xtras;           /* reserved                               */
  AP_UINT32       xtra_len;        /* reserved                               */
  unsigned char   rx_thruput_class; /* Max Rx speed of calling DTE           */
  unsigned char   tx_thruput_class; /* Max Tx speed of calling DTE           */
} VQL_LS_SPEC_DATA;
```

Port-specific data for Token Ring, Ethernet, FDDI:

```
typedef struct vdl_sap_cfg
{
  V0_MUX_INFO     mux_info;         /* Streams config info                   */
```

```
} VDL_SAP_CFG;
```

Link-specific data for Token Ring, Ethernet, FDDI:

```
typedef struct vdl_ls_cfg
{
  V0_MUX_INFO     mux_info;          /* Streams config info                */
  AP_UINT16       reserve1;          /* reserved                           */
  AP_UINT16       reserve2;          /* reserved                           */
  AP_UINT16       test_timeout;      /* TEST timeout value in seconds      */
  AP_UINT16       test_retry_limit;  /* TEST retransmission limit          */
  AP_UINT16       xid_timeout;       /* XID timeout value in seconds       */
  AP_UINT16       xid_retry_limit;   /* XID retransmission limit           */
  AP_UINT16       t1_timeout;        /* T1 timeout value in seconds        */
  AP_UINT16       t1_retry_limit;    /* I-frame retransmission limit       */
} VDL_LS_CFG;
```

Data for all DLC types:

```
typedef struct v0_mux_info
{
  AP_UINT16       dlc_type;             /* DLC implementation type         */
  unsigned char   need_vrfy_fixup;      /* reserved                        */
  unsigned char   num_mux_ids;          /* reserved                        */
  AP_UINT32       card_type;            /* reserved                        */
  AP_UINT32       adapter_number;       /* reserved                        */
  AP_UINT32       oem_data_length;      /* reserved                        */
  int             mux_ids[5];           /* reserved                        */
} V0_MUX_INFO;
```

For Token Ring, Ethernet, or FDDI, the *address* parameter in the link_address structure is replaced by the following:

```
typedef struct tr_address
{
  unsigned char   mac_address[6];       /* reserved                        */
  unsigned char   lsap_address;         /* local SAP address               */
} TR_ADDRESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_PORT

port_name

Name of port being defined. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes.

def_data.description

A null-terminated text string (0-31 characters followed by a null character) describing the port. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_PORT verb, but SNAplus2 does not make any other use of it.

def_data.initially_active

Specifies whether this port is automatically started when the node is started. Possible values are:

AP_YES

The port is automatically started when the node is started.

AP_NO

The port is automatically started only if an LS that uses it is defined to be initially active; otherwise it must be started manually.

def_data.dlc_name

Name of associated DLC. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. The specified DLC must have already been defined by a DEFINE_DLC verb.

def_data.port_type

Type of line used by the port.

For SDLC, the following values are allowed:

AP_PORT_SWITCHED

Switched line.

AP_PORT_NONSWITCHED

Nonswitched line.

For QLLC, this parameter must be set to AP_PORT_SWITCHED.

For Token Ring / Ethernet / FDDI, this parameter must be set to AP_PORT_SATF (shared access transport facility).

def_data.port_attributes

This is a one-bit parameter that may take the following values:

AP_NO

Incoming calls are resolved by CP name.

AP_RESOLVE_BY_LINK_ADDRESS

This specifies that an attempt is made to resolve incoming calls by using the link address on CONNECT_IN before using the CP name (or node ID) carried on the received XID3 to resolve them. This bit is ignored unless the *port_type* parameter is set to AP_PORT_SWITCHED.

def_data.port_number

The number of the port.

def_data.max_rcv_btu_size

Maximum BTU size that can be received. The value includes the length of the TH and RH (total 9 bytes) as well as the RU. Specify a value in the range 265-65,535 (265-4105 for SDLC).

def_data.tot_link_act_lim

Total link activation limit (the maximum number of links that can be active at any time using this port).

For an SDLC port with *port_type* set to AP_NONSWITCHED and *ls_role* set to AP_LS_PRIor AP_LS_SEC, the range is 1-256 (a value greater than 1 defines a multi-drop primary link or a multi-PU secondary link). For all other SDLC ports, this

parameter must be set to 1.

def_data.inb_link_act_lim

Inbound link activation limit (the number of links reserved for inbound activation). The sum of *inb_link_act_lim* and *out_link_act_lim* must not exceed *tot_link_act_lim*; the difference between *inb_link_act_lim* and *tot_link_act_lim* defines the maximum number of links that can be activated outbound at any time.

For an SDLC port with *port_type* set to AP_NONSWITCHED, this parameter must be zero. If *port_type* is set to AP_SWITCHED, then the port must be defined to accept either incoming calls (*inb_link_act_lim = 1* and *out_link_act_lim = 0*) or outgoing calls (*inb_link_act_lim = 0* and *out_link_act_lim = 1*).

def_data.out_link_act_lim

Outbound link activation limit (the number of links reserved for outbound activation). The sum of *inb_link_act_lim* and *out_link_act_lim* must not exceed *tot_link_act_lim*; the difference between *out_link_act_lim* and *tot_link_act_lim* defines the maximum number of links that can be activated inbound at any time.

For an SDLC port with *port_type* set to AP_NONSWITCHED, this parameter must be equal to *tot_link_act_lim*. If *port_type* is set to AP_SWITCHED, then the port must be defined to accept either incoming calls (*inb_link_act_lim = 1* and *out_link_act_lim*

= 0) or outgoing calls (*inb_link_act_lim = 0* and *out_link_act_lim = 1*).

def_data.ls_role

Link station role.

For SDLC or QLLC, the following values are allowed:

AP_LS_PRI Primary

AP_LS_SEC Secondary

AP_LS_NEG Negotiable

For Token Ring / Ethernet/ FDDI, this must be set to AP_LS_NEG.

implicit_dspu_template

Specifies the DSPU template, defined on the DEFINE_DSPU_TEMPLATE verb. This template is used for definitions if the local node is to provide PU concentration for an implicit link activated on this port. If the template specified does not exist or is already at its instance limit when the link is activated, activation will fail. This template name is an 8-byte string in a locally displayable character set.

If the *implicit_dspu_services* parameter is not set to AP_PU_CONCENTRATION, the *implicit_dspu_template*parameter is reserved.

implicit_ls_limit

Specifies the maximum number of implicit link stations that can be active on this port simultaneously, including dynamic links and links activated for Discovery. Specify a value in the range 1-65,534 or specify 0 (zero) to indicate no limit. A value of AP_NO_IMPLICIT_LINKS indicates that no implicit links are allowed.

implicit_dspu_services

Specifies the services that the local node will provide to the downstream PU across implicit links activated on this port. Possible values are:

AP_DLUR

Local node will provide DLUR services for the downstream PU (using the default DLUS configured through the DEFINE_DLUR_DEFAULTS verb.

AP_PU_CONCENTRATION

Local node will provide PU concentration for the downstream PU. It will also put in place definitions as specified by the DSPU template specified for the

parameter *implicit_dspu_template*.

AP_NONE

Local node will provide no services for this downstream PU.

def_data.implicit_deact_timer

Implicit limited resource link deactivation timer (in seconds). If *implicit_limited_resource* is set to AP_INACTIVITY, an implicit link using this port will be deactivated if no data flows on it for the time specified by this parameter.

The minimum value is 5; values in the range 1-4 will be interpreted as 5. Zero indicates no timeout (the link is not deactivated, as though *implicit_limited_resource* were set to AP_NO). This parameter is reserved if *implicit_limited_resource* is set to any value other than AP_INACTIVITY.

def_data.act_xid_exchange_limit

Activation XID exchange limit.

def_data.nonact_xid_exchange_limit

Non-activation XID exchange limit.

def_data.ls_xmit_rcv_cap

Specifies the link station transmit/receive capability. Possible values are:

AP_LS_TWS	Two-way simultaneous
AP_TWA	Two-way alternating

def_data.max_ifrm_rcvd

Maximum number of I-frames that can be received by the local link stations before an acknowledgment is sent. Range: 1-127.

def_data.target_pacing_count

Numeric value between 1 and 32,767 inclusive indicating the desired pacing window size. (The current version of SNAplus2 does not make use of this value.)

def_data.max_send_btu_size

Maximum BTU size that can be sent from this port. This value is used to negotiate the maximum BTU size that a pair of link stations can use to communicate with each other. The value includes the length of the TH and RH (total 9 bytes) as well as the RU. Specify a value in the range 265-65,535 (265-4105 for SDLC).

def_data.dlc_data.length

Length of the port address (in the following parameter).

def_data.dlc_data.address

Port address.

For SDLC, this is a 1-byte address. If *ls_role* is set to AP_LS_SEC, or if *ls_role* is set to AP_LS_NEG and the local station becomes secondary after LS role negotiation, this address is used in the response to an incoming call. If the local station is primary, or if the port is used only for outgoing calls, this parameter is reserved.

For QLLC, this is a string of 1 - 14 bytes, speciying the local X.25 DTE address of the port. This address must match the address configured in your X.25 driver for this network.

For Token Ring, Ethernet, or FDDI, this is a 7-byte string consisting of 6 zero bytes followed by a 1-byte local SAP address. (The first six bytes of the address normally contain the MAC address, but this value is used only on the LS and is reserved on the port).

def_data.dlc_data.lsap_address

Local SAP address of the port. Specify a multiple of 0x04 in the range 0x04-0xEC. The value must be specified as two hexadecimal digits preceded by 0x.

def_data.implicit_cp_cp_sess_support

Specifies whether CP-CP sessions are permitted for implicit link stations using this port. Possible values are:

AP_YES

CP-CP sessions are permitted for implicit LSs.

AP_NO

CP-CP sessions are not permitted for implicit LSs.

def_data.implicit_limited_resource

Specifies whether implicit link stations off this port should be defined as limited resources. Possible values are:

AP_NO

Implicit links are not limited resources, and will not be deactivated automatically.

AP_NO_SESSIONS

Implicit links are limited resources, and will be deactivated automatically when no active sessions are using them.

AP_INACTIVITY

Implicit links are limited resources, and will be deactivated automatically when no active sessions are using them or when no data has flowed for the time period specified by the *implicit_deact_timer* field.

def_data.implicit_hpr_support

This parameter is reserved.

def_data.implicit_link_lvl_error

This parameter is reserved.

def_data.default_tg_chars

Default TG characteristics. These are used for implicit link stations using this port, and as the default TG characteristics for defined link stations that do not have TG characteristics explicitly defined. The TG characteristics parameters are ignored if the LS is to a downstream PU.

For details of these parameters, see “DEFINE_LS”.

def_data.port_spec_data_len

Length of port-specific data. The data should be concatenated to the basic VCB structure.

def_data.link_spec_data_len

Length of link-specific data. The link-specific data should be concatenated immediately following the port-specific data.

For details of these parameters, see "DEFINE_LS"; the values specified on DEFINE_PORT are used as defaults for processing incoming calls (when the LS name is not initially known).

For SDLC, the parameters in the stub_spec_data structure within this structure are reserved.

Port-specific data for SDLC:

mux_info.dlc_type

Type of the DLC. Set this to AP_IMPL_SDLC_SL

idle_timer

Timer used to detect a completely inactive line. The line is considered idle when nothing (not even frame data that is not valid) has been received in this time. The timer is specified in milliseconds.

idle_timer_retry

Number of times to rerun the idle timer before failure. This is used in conjunction with *idle_timer* to provide the overall idle timeout period. This should be longer than either the nonproductive receive timer or the contact and disconnect timers.

A value of 0xFFFF indicates an unlimited retry count. A value of 0x0001 indicates that an outage should be generated after the first timer expiry.

np_rcv_timer

For SDLC secondary, the nonproductive receive timeout corresponds to the time allowed for receipt of a valid frame from the primary. This is usually set in conjunction with the retry limit to give a long timeout before outage (such as 30). The timer is specified in

milliseconds.

np_rcv_timer_retry

The nonproductive receive retry limit is used in conjunction with the timeout value to provide the overall time before an outage message is issued.

A value of 0xFFFF indicates an unlimited retry count. A value of 0x0001 indicates that an outage should be generated after the first timer expiry.

write_timer

The write timeout corresponds to the maximum time allowed to transmit a complete frame. This is usually set in conjunction with the retry limit to give a long timeout before outage of about 30s. The timer is specified in milliseconds.

write_timer_retry

The write timeout retry limit is used in conjunction with the timeout value to provide the overall time before an outage message is issued.

A value of 0xFFFF indicates an unlimited retry count. A value of 0x0001 indicates that an outage should be generated after the first timer expiry.

link_conn_timer

The link connection timeout together with the retry limit corresponds to the time interval after which SNAplus2 fails an attempt to activate an LS because it has not detected that DSR has been raised. The timer is specified in milliseconds.

link_conn_timer_retry

The link connection retry limit specifies the number of times to test for link connection before failing an attempt to activate an LS.

A value of 0xFFFF indicates an unlimited retry count. A value of 0x0001 indicates that an outage should be generated after the first timer expiry.

pri_fdplx

Specifies whether the primary SDLC station on this link supports full duplex. Possible values are:

AP_YES

The primary station supports full-duplex. FULL_DUPLEX in the *stub_spec_data.opt1* parameter below must also be set.

AP_NO

The primary station does not support full-duplex.

sec_fdplx

Specifies whether the secondary station(s) on the link is (are) full duplex. Possible values are:

AP_YES

The secondary station supports full-duplex. FULL_DUPLEX in the *stub_spec_data.opt1* parameter below must also be set.

AP_NO

The secondary station does not support full-duplex.

If both *pri_fdplx* and *sec_fdplx* are set, then the primary can transmit to the secondary even when the secondary has the poll bit. If *pri_fdplx* then the primary can transmit to one secondary when another secondary has the poll bit.

A secondary itself can only transmit when it has the poll bit even when the primary and secondary are full duplex.

use_rej

Specifies whether SNAplus2 can send a REJ frame on receiving an I-frame with a sequence number that is not valid on this port. (SNAplus2 always accepts a REJ frame, regardless of the setting of this parameter.) Possible values are:

AP_YES

SNAplus2 can send a REJ frame.

AP_NO

SNAplus2 cannot send a REJ frame; instead, it

requests retransmission of frames through RR, RNR or I-frames.

Use of REJ is only beneficial when full duplex protocol operation is being used. Some SDLC stations may not recognize the REJ command; to connect to them, the value AP_NO must be set.

port_type

Type of the port.

If *def_data.port_type* above is AP_NONSWITCHED, set this parameter to AP_PORT_LEASED.

For a switched line (*def_data.port_type* is AP_SWITCHED):

- If the port is used for outgoing calls (*inb_link_act_lim* is zero), set this parameter to AP_PORT_SW_DIAL (Dial-out capabilities).
- If the port is used for incoming calls (*out_link_act_lim* is zero), set this parameter to AP_PORT_SW_ANSWER (Answer capabilities).

If the switched line's dial-out or answer capabilities are manual, not automatic, combine the value above (using a logical OR) with AP_PORT_SW_MAN.

max_xid_size

The maximum size of an XID that will be sent or received on this link. This field is present to help minimize buffer usage. A safe absolute maximum is 256 bytes.

max_retry_count

The maximum number of times that a frame or group of frames may be retransmitted on this port before a problem is diagnosed. Usually set to about 5.

stub_spec_data_len

Length of the Stub specific data that follows. Set this to size of (STUB_SPEC_DATA).

stub_spec_data.opt1

HMOD port options flag 1. Set the appropriate bits of this field as follows (bit 7 is the most significant bit):

bit 7	4-wire connection (2 wire connection if not set).
bit 6	Use NRZI (NRZ if not set).
bit 5	Reserved (must be set to 0).
bit 4	Line is full-duplex (half-duplex if not set). This bit must be set if either *pri_fdplx* or *sec_fdplx* above is set to AP_YES.
bit 3	Internal line speed clocking (external if not set). This option may be ignored if the underlying SDLC hardware supports only external clocking.
bit 2	Use DMA to transfer data to and from the communications card (do not use DMA if not set).
bits 1, 0	Reserved (must be set to 0).

stub_spec_data.opt2

HMOD port options flag 2. Set the appropriate bits of this field as follows (bit 7 is the most significant bit):

bit 7	Use DSRS (do not use if not set)
bit 6	Select Standby On (Select Standby Off if not set)
bit 5	Monitor DCD (do not monitor if not set)
bit 4	Stream flags on the line (do not stream if not set)
bits 3-0	Reserved

stub_spec_data.linesp

The line speed for the line used on this port. For example, 2400 (0x00000960) for a 2400 baud line. Valid values are in the range 600-38400 baud. The exact meaning of this parameter depends on the value set on

the *physical_link* parameter.

- If *physical_link* is set to SDLC_PL_X21, then the *stub_spec_data.linesp* parameter is ignored.
- If *physical_link* is set to SDLC_PL_V25 or SDLC_PL_SMART_MODEM, then the value of the *stub_spec_data.linesp* parameter is the speed at which the dial string is sent to the modem.
- If *physical_link* is set to any other value, then the value of the *stub_spec_data.linesp* parameter is the speed of data transfer, only valid if external clocking is specified.

stub_spec_data.rcv_pool_size

The initial number of buffers reserved for receiving data on the port. Set this to the value 4.

stub_spec_data.poll_wait

The number of seconds for which the port waits between successive polls of the line for errors or for raising of DSR. A suitable value is 1 second. Increase this value for better throughput if there is only small likelihood of line errors and the line is leased.

stub_spec_data.hmod_data_len

Length of the dial data string that follows (in the *hmod_data* parameter). If no dial data is specified, set this parameter to zero.

stub_spec_data.hmod_data

Dial data for incoming calls. (The dial string used to initiate outgoing calls is specified in the LS definition.) This parameter applies only to switched links; it is reserved if the port is defined to be nonswitched.

This is an ASCII string, specifying the dial string that must be passed to the modem to instruct it to respond to incoming calls. Support for dial data depends on the SDLC adapter and modem that you are using; if they do not support dial data, set this parameter to a null string.

Port-specific data for QLLC:

mux_info.dlc_type

Type of the DLC. Set this to AP_IMPL_NLI_QLLC.

cud_mode

Specifies the type of matching required between the Call User Data (CUD) supplied on an incoming call and the *cud* parameter below. Possible values are:

VQL_DONTCARE

CUD on incoming calls is not checked.

VQL_IDENTITY

The received CUD must match the string specified in the *cud* parameter.

VQL_STARTSWITH

The initial bytes (up to *cud_len*) of the received CUD must match the string specified in the *cud* parameter; any bytes following *cud_len* are not checked.

cud_len

Specifies the length of the Call User Data (in the *cud* parameter below).

cud

Call user data to be used for verifying incoming calls. If *cud_mode* above is set to VQL_IDENTITY or VQL_STARTSWITH, incoming calls are accepted only if they specify a CUD string that matches the value defined in this parameter. If *cud_mode* is set to VQL_DONTCARE, this parameter is ignored and CUD strings on incoming calls are not checked.

add_mode

Specifies the type of matching required between the address supplied on an incoming call and the port address defined in the *address* parameter above. Possible values are:

VQL_DONTCARE

The address on incoming calls is not checked.

VQL_IDENTITY

The received address must match the string specified in the *address* parameter.

VQL_STARTSWITH

The initial bytes (up to *add_len*) of the received address must match the string specified in the *address* parameter; any bytes following *add_len* are not checked.

If the *address* parameter is set to a null string, this parameter must be set to VQL_DONTCARE.

add_len

If *add_mode* is set to VQL_STARTSWITH, this parameter specifies the number of bytes of the port address to be checked.

For example, if *add_len* is set to 2, an incoming call is accepted if the first two bytes of the address supplied on the call match the first two bytes of the *address* parameter (regardless of whether subsequent bytes match).

For other values of *add_mode*, this parameter is ignored.

Port-specific data for Token Ring, Ethernet, or FDDI:

mux_info.dlc_type

Type of the DLC.

Possible values are:

AP_IMPL_TR_DLPI

Token Ring

AP_IMPL_ETHER_DLPI

Ethernet

AP_IMPL_FDDI_DLPI

FDDI

sap_spec_data.ack_timeout

Timeout in milliseconds within whichSNAplus2 expects a response to any I-frames sent to the adjacent link station.

sap_spec_data.p_bit_timeout

Time in milliseconds that SNAplus2 waits for a response to a frame sent with the POLL bit set.

sap_spec_data.t2_timeout

Period in milliseconds that SNAplus2 will withhold a response to a received I-frame to allow further I-frames to be received and acknowledged with the same RR, thus reducing acknowledgment traffic. At the latest, SNAplus2 will send the acknowledgment after this period, but may send the acknowledgment before this period is over.

sap_spec_data.rej_timeout

Time in seconds during which SNAplus2 expects to receive a response to an REJ frame.

sap_spec_data.busy_state_timeout

Time in seconds that SNAplus2 waits for indication of clearance of a busy condition at the adjacent link station.

sap_spec_data.idle_timeout

RR keep-alive interval in seconds for an otherwise idle link.

sap_spec_data.max_retry

Maximum permitted number of retries when waiting for any response or busy state to clear.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PORT_NAME

The *port_name* parameter was not valid.

AP_INVALID_DLC_NAME

The specified *dlc_name* did not match any defined DLC.

AP_INVALID_PORT_TYPE

The *port_type* parameter was not set to a valid value.

AP_INVALID_BTU_SIZE

The *max_rcv_btu_size* parameter was not set to a valid value.

AP_INVALID_LS_ROLE

The *ls_role* parameter was not set to a valid value.

AP_INVALID_LINK_ACTIVE_LIMIT

One of the activation limit parameters was not set to a valid value.

AP_INVALID_MAX_IFRM_RCVD

The *max_ifrm_rcvd* parameter was not set to a valid value.

AP_INVALID_HPR_SUPPORTED

A reserved parameter was not set to zero.

AP_INVALID_LS_ROLE

The *ls_role* parameter was not set to a valid value.

AP_INVALID_DSPU_SERVICES

The *implicit_dspu_services* parameter was not set to a valid value.

AP_PU_CONC_NOT_SUPPORTED

The *implicit_dspu_services* parameter was set to a reserved value.

AP_INVALID_TEMPLATE_NAME

The DSPU template specified on the *implicit_dspu_template* parameter was not valid.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_PORT_ACTIVE

The specified port cannot be modified because it is currently active.

AP_DUPLICATE_PORT_NUMBER

A port with the specified *port_number* has already been defined.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Incoming Calls

If you are configuring a port that will accept incoming calls (as defined by the *tot_link_act_lim*, *inb_link_act_lim*, and *out_link_act_lim* parameters), there is generally no need to define an LS to be used for

these calls, because SNAplus2 will define one dynamically when the incoming call is received. However, if the incoming calls are from a host computer that supports dependent LUs or from a downstream computer using PU concentration, you need to define an LS explicitly, because the LS definition includes the name of the PU associated with the dependent LUs.

When an incoming call arrives at the port, SNAplus2 checks the address specified on the call against the addresses specified for LSs defined on the port (if any), to determine if an LS has already been defined for the call. If the address does not match, an LS is defined dynamically. To ensure that the explicit LS definition (including the required PU name) is used, ensure that the address defined for this LS matches the address that will be supplied by the host or the downstream computer on the incoming call. For Token Ring / Ethernet / FDDI, both the MAC and SAP addresses must match in order to select the correct LS.

DEFINE_RCF_ACCESS

DEFINE_RCF_ACCESS specifies access to the SNAplus2 Remote Command Facility (RCF): the user ID used to run HP-UX Command Facility (UCF) commands, and the restrictions on which administration commands can be issued using the Service Point Command Facility (SPCF). For more information about SPCF and UCF, see the *HP-UX SNAplus2 Administration Guide*. You can use this verb to permit access to both SPCF and UCF, or to only one of them.

This verb must be issued to the domain configuration file; it can be used to specify the RCF access for the first time, or to modify an existing definition. SNAplus2 acts on these parameters during node startup; if these parameters are changed while a node is running, the changes do not take effect on the server where the node is running until the node is stopped and restarted.

VCB Structure

```
typedef struct define_rcf_access
{
    AP_UINT16        opcode;                /* Verb operation code          */
    unsigned char    reserv2;               /* reserved                     */
    unsigned char    format;                /* reserved                     */
    AP_UINT16        primary_rc;            /* primary return code          */
    AP_UINT32        secondary_rc;          /* secondary return code        */
    unsigned char    ucf_username[32];      /* UCF username                 */
    AP_UINT32        spcf_permissions;      /* SPCF permissions             */
} DEFINE_RCF_ACCESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_RCF_ACCESS

ucf_username

Specifies the HP-UX user name of the UCF user. This parameter is a null-terminated ASCII string. Do not

specify the name root, because SNAplus2 does not allow UCF commands to be run as root for security reasons.

All UCF commands will be run using this user's user ID, with the default shell, default group ID, and access permissions that are defined on the HP-UX system for this user.

To prohibit access to UCF, set this parameter to a null string.

spcf_permissions

Specifies the types of SNAplus2 verbs that can be accessed using SPCF. Set this to AP_NONE to prevent access to SPCF, or to one or more of the following values (combined using a logical OR):

AP_ALLOW_QUERY_LOCAL

QUERY_* verbs are permitted.

AP_ALLOW_DEFINE_LOCAL

DEFINE_*, SET_*, DELETE_*, ADD_*, and REMOVE_* verbs, and also INIT_NODE, are permitted.

AP_ALLOW_ACTION_LOCAL

“Action” verbs are permitted: START_*, STOP_*, ACTIVATE_*, DEACTIVATE_*, and also APING, INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, and RESET_SESSION_LIMIT.

AP_ALLOW_QUERY_REMOTE

The QUERY_* verbs are allowed to be directed at any node in the domain.

AP_ALLOW_DEFINE_REMOTE

The DEFINE_*, SET_*, DELETE_*, ADD_*, REMOVE_*, and INIT_NODE verbs are allowed to be directed at any node in the domain.

AP_ALLOW_ACTION_REMOTE

The START_*, STOP_*, ACTIVATE_*, DEACTIVATE_*, APING, INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, and RESET_SESSION_LIMIT verbs are allowed to be directed at any node in the domain.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_UCF_USER_CANNOT_BE_ROOT

The *ucf_username* parameter specified the name root, which is not allowed.

AP_INVALID_SPCF_SECURITY

The *spcf_permissions* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_RJE_WKSTN

DEFINE_RJE_WKSTN defines a workstation for use by a group of SNAplus2 RJE users.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct de
fine_rje_wkstn
{
    AP_UINT16           opcode;               /* verb operation code         */
    unsigned char       reserv2;              /* reserved                    */
    unsigned char       format;               /* reserved                    */
    AP_UINT16           primary_rc;           /* primary return code         */
    AP_UINT32           secondary_rc;         /* secondary return code       */
    unsigned char       workstation_name[4];  /* workstation name            */
    RJE_WKSTN_DEF_DATA  def_data;
} DEFINE_RJE_WKSTN;


typedef struct rje_wkstn_def_data
{
    unsigned char       description[32];      /* Description - null          */
                                              /*  terminated                 */
    unsigned char       reserv1[16];          /* reserved                    */
    unsigned char       primary_user[32];     /* primary user name           */
    unsigned char       group_name[32];       /* group name                  */
    unsigned char       system_name[64];      /* computer where workstation  */
                                              /*  runs                       */
    AP_UINT16           num_lus;              /* number of LUs used by wkstn */
    unsigned char       lu_name[5][8];        /* names of LUs                */
} RJE_WKSTN_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_RJE_WKSTN

workstation_name

The name of the workstation. This is an ASCII string of 1-4 characters; it is not case-sensitive.

def_data.description

An optional text string (0-31 characters followed by a null character). The string is for information only; it is stored in the configuration file and returned on the QUERY_RJE_WORKSTATION verb, but SNAplus2 does not make use of it.

def_data.primary_user

The HP-UX user ID of the main user of this workstation. This is an ASCII string, padded on the right with spaces.

This must be a valid user ID on the HP-UX computer specified in *system_name* below, and must be in the group specified for *group_name* below. The files and directories that SNAplus2 creates for use by this workstation will be owned by this user ID.

def_data.group_name

The HP-UX group name of the group of HP-UX users who use this workstation. This is an ASCII string, padded on the right with spaces.

This must be a valid group name on the HP-UX computer specified in *system_name* below, and all users who will use the workstation must be members of the group. The files and directories which SNAplus2 creates for use by this workstation will be owned by this group ID.

def_data.system_name

The name of the HP-UX server or client on which this workstation will run. This is an ASCII string, padded on the right with spaces.

Before the workstation can be used, you must create the workstation directory `/var/opt/sna/rje/WKST` on the specified computer; `WKST` represents the name of the RJE workstation, as specified by the *workstation_name* parameter above. You should also create a workstation style file with the name `WKST.sty`

in the directory `/etc/opt/sna`; for more information about the workstation style file, refer to the *HP-UX SNAplus2 RJE Users Guide*. All other files and directories required by the workstation will be created automatically when the workstation is started.

In general, RJE users must run the spool commands (to submit, list, or cancel jobs) on the same computer as the workstation. If you need to relax this restriction, you can do this by sharing the workstation directory (and its subdirectories) using NFS; the spool commands can then be issued from any computer that has this shared directory mounted.

If you want to allow the workstation to be run on more than one computer, set this parameter to 64 binary zeros. In this case, you need to be aware of the following:

- Any computer where the workstation may be run must have the workstation directory and style file set up as described above, and must include the workstation's primary user and group in the HP-UX user and group setup. You are recommended to set up the workstation directory on one computer and share it using NFS, as described above.
- Jobs may be submitted to the workstation on any computer where the workstation may be run, but will not be sent to the host until the workstation is started on the computer where the jobs are spooled.
- Output returned from the host will be sent to the appropriate file, directory, or program on the computer where the workstation is running when it receives the output; this is not necessarily the same computer from which the job was originally submitted.

def_data.num_lus

Number of LUs used by the workstation (1-5). This must match the number of LU names specified in the following parameter.

def_data.lu_name

Names of the LUs that this workstation uses. Each name is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces. It must match the name of a type 0-3 LU that has already been defined.

If the workstation uses more than one LU, all LUs must be defined at the same host (although they may be associated with different local PUs and so access the host using different LSs).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LU_NAME

The *lu_name* parameter did not match the name of any defined type 0-3 LU or LU pool.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_SECURITY_ACCESS_LIST

DEFINE_SECURITY_ACCESS_LIST defines a list of users who can access a particular local LU or invokable TP, so that access to that LU or TP is restricted to the named users. It can also be used to add user names to an existing security access list. The user names in the list are defined using the DEFINE_USERID_PASSWORD verb.

To restrict access for a particular local LU or invokable TP, you need to do the following:

- Ensure that each authorized user of the LU or TP is defined using the DEFINE_USERID_PASSWORD verb.
- Use the DEFINE_SECURITY_ACCESS_LIST verb to define a security access list containing all of these user IDs.
- Specify the name of this security access list on the DEFINE_LOCAL_LU or DEFINE_TP verb that defines the LU or TP.

When an incoming Allocate request arrives for a local LU or an invokable TP that has a security access list defined, the invoking application must indicate that conversation security is to be used, and specify a user ID. In addition to the standard conversation security checking (against user IDs specified using the DEFINE_USERID_PASSWORD verb), SNAplus2 checks the user ID in the incoming allocate request against the security access list defined for the LU or TP, and rejects the conversation if the user ID does not match. If both the LU and the TP have security access lists defined, the user ID must be in both lists.

If a local LU or an invokable TP does not have a security access list defined, but is still configured to require conversation security, the standard conversation security checking still applies.

VCB Structure

The DEFINE_SECURITY_ACCESS_LIST verb contains a variable number of `security_user_data` structures; these define the user names to be added to the security access list.

The user name structures are included at the end of the `def_data` structure; the nmber of these structures is specified by the *`num_users`* parameter.

```
typedef struct define_security_access_list
{
   AP_UINT16         opcode;                /* verb operation cod         */
   unsigned char     reserv2;               /* reserved                   */
   unsigned char     format;                /* reserved                   */
   AP_UINT16         primary_rc;            /* primary return code        */
   AP_UINT32         secondary_rc;          /* secondary return code      */
   unsigned char     list_name[14];         /* name of this list          */
   unsigned char     reserv3[2];            /* reserved                   */
   SECURITY_LIST_DEF def_data;              /* security access list       */
} DEFINE_SECURITY_ACCESS_LIST;

typedef struct security_list_def
{
   unsigned char     description[32];      /* description                */
   unsigned char     reserv3[16];          /* reserved                   */
   AP_UINT32         num_users;            /* number of users being added */
   unsigned char     reserv2[16];          /* reserved                   */
} SECURITY_LIST_DEF;

typedef struct security_user_data
{
   AP_UINT16         sub_overlay_size;     /* reserved                   */
   unsigned char     user_name[10];        /* user name                  */
} SECURITY_USER_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEVINE_SECURITY_ACCESS_LIST

list_name

Name of the security access list. This is an ASCII string, padded on the right with spaces.

If this name matches an existing security access list, the users defined by this verb are added to the list; otherwise, a new list is created.

def_data.description

A null-terminated text string (0 - 31 characters followed by a null character) describing the security

access list. This string is for information only. It is stored in the node's configuration file and returned on the QUERY_SECURITY_ACCESS_LIST verb, but SNAplus2 does not make any other use of it.

def_data.num_users

Number of user names being defined by this verb. Each user must be specified by a security_user_data structure following the def_data structure.

For each user name in the list, up to the number specified in *num_users*, a security_user_data structure is required with the following information:

user_name Name of the user. This is a user ID defined using the DEFINE_USERID_PASSWORD verb.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LIST_NAME

The *list_name* parameter contained a character that was not valid.

AP_INVALID_USER_NAME

One or more of the specified user names was not valid.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_TN3270_ACCESS

DEFINE_TN3270_ACCESS defines TN3270 access details for a particular client (or default TN3270 access details for all clients). Each verb specified details for one or more sessions. Each session is uniquely identified by the client address and the server port number. The DEFINE_TN3270_ACCESS verb can be used to define a new user, to define new sessions for use by an existing client, or to modify the session parameters. (To delete sessions from an existing user, use DELETE_TN3270_ACCESS.)

VCB Structure

The DEFINE_TN3270_ACCESS verb contains a variable number of `tn3270_session_def_data` structures; these define the user's sessions. The session structures are included at the end of the `def_data` structure; the number of these structures is specified by the *`num_sessions`* parameter.

```
typedef struct define_tn3270_access
{
   AP_UINT16           opcode;             /* verb operation code            */
   unsigned char       reserv2;            /* reserved                       */
   unsigned char       format;             /* reserved                       */
   AP_UINT16           primary_rc;         /* primary return code            */
   AP_UINT32           secondary_rc;       /* secondary return code          */
   AP_UINT16           default_record;     /* is this the DEFAULT record?    */
   unsigned char       client_address[68]; /* address of TN3270 user         */
   TN3270_ACCESS_DEF_DATA def_data;
} DEFINE_TN3270_ACCESS;

typedef struct tn3270_access_def_data
{
   unsigned char       description[32];    /* Description - null terminated  */
   unsigned char       reserv1[16];        /* reserved                       */
   AP_UINT16           address_format;     /* Format of client address       */
   AP_UINT32           num_sessions;       /* Number of sessions being added */
   unsigned char       reserv3[12];        /* reserved                       */
} TN3270_ACCESS_DEF_DATA;

typedef struct tn3270_session_def_data
{
```

```
   AP_UINT16           sub_overlay_size;   /* reserved                       */
   unsigned char       description[32];    /* Session description            */
   unsigned char       tn3270_support;     /* Level of TN3270 support        */
   unsigned char       allow_specific_lu;  /* Allow access to specific LUs   */
   unsigned char       printer_lu_name[8]; /* Generic printer LU/pool        */
                                           /*   accessed                     */
   unsigned char       reserv1[6];         /* reserved                       */
   AP_UINT16           port_number;        /* TCP/IP port used to access     */
                                           /*  server                        */
   unsigned char       lu_name[8];         /* Generic display LU/pool        */
                                           /*  accessed                      */
   unsigned char       session_type;       /* Unused in SNAplus2 V6          */
   unsigned char       model_override;     /* Unused in SNAplus2 V6          */
   unsigned char       reserv3[4];         /* reserved                       */
   AP_UINT32           reserv4;            /* reserved                       */
} TN3270_SESSION_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_TN3270_ACCESS

sub_overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

default_record

Specifies whether this verb defines a default record, which will be used by any TN3270 user not explicitly identified by a TCP/IP address. If a TN3270 user attempts to contact the TN server node, and the user's TCP/IP address does not match any DEFINE_TN3270_ACCESS record in the configuration but there is a default record defined, the parameters from this record will be used. Possible values are:

AP_YES

This verb defines a default record. The

client_address and *address_format* parameters are reserved.

AP_NO

This verb defines a normal TN3270 user record.

A default record provides access to the TN server function for any TN3270 user that can determine the TCP/IP address of the computer where the TN server is running. To restrict the use of TN server to a specific group of users, either do not include the default record, or leave it with no 3270 LU or LU pool configured so that it cannot be used.

You can also set up a default record for most users, but explicitly exclude one or more TCP/IP addresses. To do this, define the addresses to be excluded as TN server users, and leave them with no 3270 LU or LU pool configured.

client_address

The TCP/IP address of the computer on which the TN3270 program runs. This is a string of 1-64 characters; the remaining bytes in the string should be set to nulls.

The address can be specified as a dotted-decimal IP address (such as 193.1.11.100), as a name (such as newbox.this.co.uk), or as an alias (such as newbox); the format is indicated by the *address_format* parameter.

If you use a name or alias, the following restrictions apply:

- The HP-UX system must be able to resolve the name or alias to a fully qualified name (either using the local TCP/IP configuration or using a Domain Name server).
- Each name or alias must expand to a unique fully qualified name; you should not configure two names for users of the same TN server node that will be resolved to the same fully qualified name.

def_data.description

An optional text string (0-31 characters followed by a null character). The string is for information only; it is stored in the configuration file and returned on a query_tn3270_access_def structure for a QUERY_TN3270_ACCESS_DEF verb, but SNAplus2 does not make use of it. You can use it to store additional information to help distinguish between users.

def_data.address_format

Specifies the format of the *client_address* parameter. Possible values are:

AP_ADDRESS_IP

IP address

AP_ADDRESS_FQN

Alias or fully qualified name

def_data.num_sessions

The number of sessions being defined or modified by this verb. Each TN3270 user may access the same TN server node with multiple sessions, by using a different TCP/IP port for each session. Each session must be specified by a tn3270_session_def_data structure following the tn3270_access_def_data structure.

For each session, a tn3270_session_data structure is required with the following information:

description

An optional text string (0-31 characters followed by a null character). The string is for information only; it is stored in the configuration file and returned on aquery_tn3270_access_def structure for a QUERY_TN3270_ACCESS_DEF verb, but SNAplus2 does not make use of it.

tn3270_support

Specifies the level of TN3270 support. Possible values are:

AP_TN3270

Specifies that TN3270E protocols are disabled.

AP_TN3270E

Specifies that TN3270E protocols are enabled. This is the default value.

TN3270 and TN3287 protocols are always enabled.

allow_specific_lu

Indicates whether access to specific LUs is allowed. Possible values are:

AP_YES

Access to specific LUs is allowed.

AP_NO

Access to specific LUs is not allowed.

printer_lu_name

Name of the printer LU or LU pool that this session uses for connections requesting a generic printer LU. This is an EBCDIC string padded on the right with EBCDIC spaces. It must match the name of a LU type 0-3 printer LU defined on this node, or an LU pool containing LUs on this node.

If a single printer LU is specified, this printer LU should not be associated with any display LU by the DEFINE_TN3270_ASSOCIATION verb. If a printer LU pool is specified, none of the printer LUs in the pool should be associated with display LUs. Allowing a single LU to be accessed as both a generic printer LU and as an associated printer LU may result in the LU not being available as an associated printer LU because it is already in use. (These rules are not enforced by the NOF API.)

This field has no affect on specific printer LU sessions.

port_number

The number of the server TCP/IP port that the TN3270 program uses to access the TN server node. If the port number matches an existing port number defined for one of this TN3270 user's sessions, the information for

that session is replaced; otherwise a new session is added.

If the TN3270 program uses TCP/IP port number 23 (the port number used by the Telnet daemon program on the HP-UX computer), you will need to set up an additional initialization file to share this port number between TN server and the Telnet daemon program. For more information, see “Using the Telnet Daemon's TCP/IP Port”.

lu_name

Name of the LU or LU pool that this session uses for connections requesting a generic display LU. This is an EBCDIC string padded on the right with EBCDIC spaces. It must match the name of a type 0-3 display LU defined on this node, or an LU pool containing LUs on this node.

If you specify an LU name, a TN3270 program with the specified TCP/IP address will be able to use only one session at a time by connecting to the specified server port number on this TN server node. If you specify an LU pool, the program can use multiple generic display LU sessions (or multiple copies of the program can access generic display LU sessions using this TN server), up to the number of LUs on this node that are available from the pool.

This parameter has no effect on specific display LU sessions.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	Possible values are:
	AP_UNKNOWN_CLIENT_ADDRESS
	The specified name or alias could not be mapped to a fully qualified name.
	AP_CLIENT_ADDRESS_CLASH
	The fully qualified name, resolved from the *client_address* parameter, clashes with one that has already been defined.
	AP_TCPIP_PORT_IN_USE
	The TCP/IP port number cannot be used by TN server because it is already in use by a different program.
	AP_INVALID_TN3270_SUPPORT
	The *tn_3270_support* parameter for one or more sessions was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Using the Telnet Daemon's TCP/IP Port

If you are setting up TN server for use with a TN3270 program that uses TCP/IP port number 23, you will need to set up the HP-UX computer where the node owning this TN server runs, to share this port number between TN server and the Telnet daemon program. To do this, take the following steps:

Step 1. Ensure that the SNAplus2 software is stopped on the HP-UX computer.

Step 2. Log on to the HP-UX computer as root.

Step 3. Edit the file /etc/inetd.conf, and find the line beginning with telnet. Make a note of the executable name and any supplied parameters for the

Telnet daemon program, which are included in this line; typically these are `/etc/telnetd` and `telnetd`. Comment out this line by inserting a # character at the start of the line, and save the file.

Step 4. Create an ASCII text file `/etc/snapinetd.conf`. This file should consist of a single line containing the Telnet daemon executable name and parameters, as determined in step 3, for example:

```
/etc/telnetd telnetd
```

Step 5. Use the HP-UX `ps` command to find the process ID of the Internet daemon program `inetd`.

Step 6. Use the HP-UX `kill` command to stop this process, by issuing the following command:

```
kill
processid
```

`processid` is the process ID that you found in step 5.

Step 7. Start the SNAplus2 Internet daemon program, by issuing the following command:

```
snapinetd
```

Step 8. Restart the Internet daemon program, by issuing the following command:

```
inetd
```

Step 9. Restart the SNAplus2 software, and then restart the node.

Steps 5, 6, 7, and 8 must be repeated each time you restart the HP-UX computer. You may want to set up a shell script containing these commands, so that it can be run at startup.

DEFINE_TN3270_ASSOCIATION

DEFINE_TN3270_ASSOCIATION defines an association between a display LU and a printer LU. This association allows a TN3270E client to connect to the printer LU that is associated with a display LU without knowing the name of the printer LU. The DEFINE_TN3270_ASSOCIATION verb can be used to define a new association or to overwrite an existing association for a particular display LU.

VCB Structure

```
typedef struct define_tn3270_association
{
   AP_UINT16        opcode;               /* verb operation code           */
   unsigned char    reserv2;              /* reserved                      */
   unsigned char    format;               /* reserved                      */
   AP_UINT16        primary_rc;           /* primary return code           */
   AP_UINT32        secondary_rc;         /* secondary return code         */
   unsigned char    display_lu_name[8];   /* Display LU name               */
   TN3270_ASSOCIATION_DEF_DATA def_data;  /* association definition        */
} DEFINE_TN3270_ASSOCIATION;

typedef struct tn3270_association_def_data
{
   unsigned char    description[32];      /* description                   */
   unsigned char    reserve0[16];         /* reserved                      */
   unsigned char    printer_lu_name[8];   /* Printer LU name               */
   unsigned char    reserv2[8];           /* reserved                      */
} TN3270_ASSOCIATION_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_TN3270_ASSOCIATION

display_lu_name

Name of the display LU to be associated with the printer that was specified by the

def_data.printer_lu_name parameter. This is an EBCDIC string padded on the right with EBCDIC spaces.

The specified display LU should be a display LU defined on the local node, but this is not enforced by the NOF API.

def_data.description

Description of the association being defined. This parameter is optional.

def_data.printer_lu_name

Name of the printer LU to be associated with the display LU that was specified by the *display_lu_name* parameter. This is an EBCDIC string padded on the right with EBCDIC spaces.

The specified printer LU should be a printer LU defined on the local node.

It is not possible for a single printer LU to be shared by two TN3270E emulators; no two TN3270 associations can specify the same printer LU.

The printer LU should not be accessible as a generic printer LU; otherwise it may not be available as an associated printer LU because it is already in use. Therefore, the associated printer LU should not be configured (directly or indirectly as a member of an LU pool) as the *printer_lu_name* in a DEFINE_TN3270_ACCESS verb.

(These rules are not enforced by the NOF API.)

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

Either the supplied display LU name or the supplied printer LU name was not a valid EBCDIC string.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_TN3270_DEFAULTS

DEFINE_TN3270_DEFAULTS defines TN3270 parameters used on all client sessions.

VCB Structure

```
typedef struct define_tn3270_defaults
{
   AP_UINT16        opcode;               /* verb operation code            */
   unsigned char    reserv2;              /* reserved                       */
   unsigned char    format;               /* reserved                       */
   AP_UINT16        primary_rc;           /* primary return code            */
   AP_UINT32        secondary_rc;         /* secondary return code          */
   TN3270_DEFAULTS_DEF_DATA def_data;     /* TN3270 defaults                */
} DEFINE_TN3270_DEFAULTS;


typedef struct tn3270_defaults_def_data
{
   unsigned char    force_responses;      /* force printer responses?       */
   unsigned char    reserv1[2];           /* reserved                       */
   unsigned char    keepalive_method;     /* method for sending keep-alives */
   AP_UINT32        keepalive_interval;   /* interval between keep-alives   */
   unsigned char    reserv2[32];          /* reserved                       */
} TN3270_DEFAULTS_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_TN3270_DEFAULTS

def_data.force_responses

Controls client responses on printer sessions. Possible values are:

AP_YES

Always request definite responses from the client printer sessions. Some 3270 emulators are unable to

print large jobs if definite responses are not requested. If necessary, set *force_responses* to AP_YES to avoid problems.

AP_NO

Request responses matching SNA traffic.

def_data.keepalive_method

Method for sending keep-alive messages. Keep-alive messages are sent to ensure periodic traffic on each connection, so that connection failure is detected. If keep-alive messages are not sent, a connection appears to be active even if a client or network fails (for example, if a client switched off). Keeping the connection status as active under these conditions wastes server resources and prevents LUs from being used for other sessions. Possible values are:

AP_NONE

Do not send keep-alive messages. This value is the default value.

AP_TN3270_NOP

Send Telnet NOP messages.

AP_TN3270_TM

Send Telnet DO TIMING-MARK messages.

def_data.keepalive_interval

Interval (in seconds) between consecutive keep-alive messages. The interval should be long enough to minimize network traffic, especially if there are typically many idle client connections. Typical values are in the range 600-7200 (10 minutes to 2 hours). The value 0 (zero) is not valid when the *keepalive_method* parameter is set to AP_TN3270_NOP or AP_TN3270_TM.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_KEEPALIVE

The *keepalive_method* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_TP

The DEFINE_TP verb provides information that SNAplus2 needs to start a TP as a result of an incoming attach from a partner LU. This verb can also be used to modify one or more fields on a previously defined TP.

The standard parameters for invoked TPs are defined in the invokable TP information file (for more information, see the *HP-UX SNAplus2 Administration Guide*). DEFINE_TP is required only if you need to specify additional parameters that cannot be set in the file: to restrict the TP to use particular options for conversation security, confirm synchronization, or conversation type (mapped or basic), or to restrict the number of instances of the TP that can be running at any time.

VCB Structure

```
typedef struct define_tp
{
   AP_UINT16        opcode;                /* verb operation code           */
   unsigned char    reserv2;               /* reserved                      */
   unsigned char    format;                /* reserved                      */
   AP_UINT16        primary_rc;            /* primary return code           */
   AP_UINT32        secondary_rc;          /* secondary return code         */
   unsigned char    tp_name[64];           /* TP name                       */
   TP_CHARS         tp_chars;              /* TP characteristics            */
} DEFINE_TP;


typedef struct tp_chars
{
   unsigned char    description[32];        /* resource description         */
   unsigned char    security_list_name[14]; /* security access list name    */
   unsigned char    reserv1[2];            /* reserved                      */
   unsigned char    conv_type;             /* conversation type             */
   unsigned char    security_rqd;          /* security support              */
   unsigned char    sync_level;            /* synchronisation level support */
   unsigned char    dynamic_load;          /* dynamic load (AP_YES)         */
   unsigned char    enabled;               /* is the TP enabled?            */
   unsigned char    pip_allowed;           /* program initialization        */
                                           /*  parameters supported         */
   unsigned char    reserv3[9];            /* reserved                      */
   AP_UINT16        tp_instance_limit;     /* limit on currently active TP  */
                                           /*  instances                    */
```

```
   AP_UINT16        tp_data_len;            /* reserved                    */
   unsigned char    tp_data[120];           /* reserved                    */
} TP_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_TP

tp_name

Name of the TP being defined.

tp_chars.description

A null-terminated text string (0-31 characters followed by a null character) describing the TP. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_TP_DEFINITION and QUERY_TP verbs, but SNAplus2 does not make any other use of it.

tp_chars.security_list_name

Name of the security access list used by this TP (defined using the DEFINE_SECURITY_ACCESS_LIST verb). This parameter restricts the use of this TP to the users named in the specified list. Only the users named in the specified list can allocate conversations with it. If you specifiy a security access list, the *tp_chars.security_rqd* parameter must be set to AP_YES.

To specify that the TP is available for use by any user, set this parameter to 14 binary zeros.

tp_chars.conv_type

Specifies the type(s) of conversation supported by this TP. Possible values are:

AP_BASIC

The TP supports only basic conversations.

AP_MAPPED

The TP supports only mapped conversations.

AP_EITHER

The TP supports either basic or mapped conversations.

tp_chars.security_rqd

Specifies whether conversation security information is required to start the TP. Possible values are:

AP_YES

A user ID and password are required to start the TP.

AP_NO

No security information is required.

tp_chars.sync_level

Specifies the values of synchronization level supported by the TP. Possible values are:

AP_NONE

The TP supports only *sync_level* NONE.

AP_CONFIRM_SYNC_LEVEL

The TP supports only *sync_level* CONFIRM.

AP_EITHER

The TP supports either *sync_level* NONE or CONFIRM.

AP_SYNCPT_REQUIRED

The TP supports only *sync_level* SYNCPT (syncpoint is required).

AP_SYNCPT_NEGOTIABLE

The TP supports any of the three *sync_level* values NONE, CONFIRM, and SYNCPT.

tp_chars.dynamic_load

This parameter must be set to AP_YES.

tp_chars.enabled

Specifies whether the TP can be attached successfully.

Possible values are:

AP_YES

TP can be attached.

AP_NO

TP cannot be attached.

tp_chars.pip_allowed

Specifies whether the TP can receive Program Initialization Parameters (PIP). Possible values are:

AP_YES

TP can receive PIP.

AP_NO

TP cannot receive PIP.

tp_chars.tp_instance_limit

Limit on the number of instances of this TP that can be active at any one time. A value of zero means no limit.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_SYSTEM_TP_CANT_BE_CHANGED

The specified TP name is the name of a TP used internally by SNAplus2 you cannot define or modify a TP with this name.

AP_INVALID_CONV_TYPE

The *conv_type* parameter was not set to a valid value.

AP_INVALID_SYNC_LEVEL

The *sync_level* parameter was not set to a valid value.

AP_INVALID_DYNAMIC_LOAD

The *dynamic_load* parameter was not set to a valid value.

AP_INVALID_ENABLED

The *enabled* parameter was not set to a valid value.

AP_INVALID_PIP_ALLOWED

The *pip_allowed* parameter was not set to a valid value.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_TP_LOAD_INFO

DEFINE_TP_LOAD_INFO defines or changes an entry that describes information to be used when a transaction program is loaded. An application must issue OPEN_FILE with a requested role of AP_TP_LOAD_INFO before issuing the DEFINE_TP_LOAD_INFO verb.

VCB Structure

```
typedef struct define_tp_load_info
{
      AP_UINT16             opcode;             /* verb operation code        */
      unsigned char         reserv2;            /* reserved                   */
      unsigned char         format;             /* reserved                   */
      AP_UINT16             primary_rc;         /* primary return code        */
      AP_UINT32             secondary_rc;       /* secondary return code      */
      unsigned char         tp_name[64];        /* TP name                    */
      unsigned char         lu_alias[8];        /* LU alias                   */
      TP_LOAD_INFO_DEF_DATA  def_data;          /* defined data               */
} DEFINE_TP_LOAD_INFO;

typedef struct tp_load_info_def_data
{
      unsigned char         description[32];     /* Description               */
      unsigned char         reserv1[16];        /* reserved                   */
      unsigned char         user_id[64];        /* User ID                    */
      unsigned char         group_id[64];       /* Group ID                   */
      AP_UINT32             timeout;            /* Timeout value              */
      unsigned char         type;               /* TP type                    */
      AP_UINT16             ltv_length;         /* Length of LTV data         */
} TP_LOAD_INFO_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_TP_LOAD_INFO

tp_name

The TP name of the TP load info entry to be defined. This is a 64-byte EBCDIC string, padded on the right with spaces if the name is shorter than 64 characters.

lu_alias

The LU alias of the TP load info entry to be defined. This is an 8-byte ASCII character string.

def_data.description

A null-terminated text string (0-32 characters followed by a null character) describing the TP load info. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_TP verb, but SNAplus2 does not make any other use of it.

def_data.user_id

User ID required to access and run the TP.

def_data.group_id

Group ID required to access and run the TP.

def_data.timeout

Timeout in seconds after the TP is loaded.

def_data.type

Specifies the TP type. The default value is AP_TP_TYPE_QUEUED. Possible values are:

AP_TP_TYPE_QUEUED

AP_TP_TYPE_QUEUED_BROADCAST

AP_TP_TYPE_NON_QUEUED

def_data.ltv_length

Length of the block of LTV data that is appended to this verb. Each LTV structure is specified in TP_LOAD_INFO_LTV.

TP_LOAD_INFO_LTV

The LTV data is specified as a series of non-byte-aligned LTVs each of which consists of the following:

- A 2-byte length field with a maximum value of 258 bytes. This field is in line format and is read or written using NB_PUT_SHORT or NB_GET_SHORT.
- A 1-byte type field with the following possible values defined with #define:

0x01	AP_TYPE_TP_PATH
0x02	AP_TYPE_TP_ARGUMENTS
0x03	AP_TYPE_TP_STDIN
0x04	AP_TYPE_TP_STDOUT
0x05	AP_TYPE_TP_STDERR
0x06	AP_TYPE_TP_ENV

- A value field consisting of up to 255 bytes of ASCII data.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_TP_TYPE

The *type* parameter was not set to a valid value.

AP_INVALID_LTV_LENGTH

An LTV *length* parameter was not set to a valid value.

AP_INVALID_LTV_TYPE

The LTV *type* parameter was not set to a valid value.

AP_INVALID_LTV_VALUE

An LTV *value* parameter contained data that was not valid.

AP_INVALID_TP_NAME

The TP *name* parameter contains EBCDIC spaces.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DEFINE_USERID_PASSWORD

DEFINE_USERID_PASSWORD defines a user ID / password pair for use with APPC and CPI-C conversation security, or adds profiles for a defined user ID and password.

VCB Structure

```
typedef struct define_userid_password
{
    AP_UINT16              opcode;            /* verb operation code        */
    unsigned char          reserv2;           /* reserved                   */
    unsigned char          format;            /* reserved                   */
    AP_UINT16              primary_rc;        /* primary return code        */
    AP_UINT32              secondary_rc;      /* secondary return code      */
    AP_UINT16              define_type;       /* what the define type is    */
    unsigned char          user_id[10];       /* user id                    */
    unsigned char          reserv3[8];        /* reserved                   */
    USERID_PASSWORD_CHARS  password_chars;    /* password characteristics   */
} DEFINE_USERID_PASSWORD;

typedef struct userid_password_chars
{
    unsigned char          description[32];   /* resource description       */
    unsigned char          reserv2[16];       /* reserved                   */
    AP_UINT16              profile_count;     /* number of profiles         */
    AP_UINT16              reserv1;           /* reserved                   */
    unsigned char          password[10];      /* password                   */
    unsigned char          profiles[10][10];  /* profiles                   */
} USERID_PASSWORD_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DEFINE_USERID_PASSWORD

define_type

Specifies how this verb is being used. Possible values

are:

AP_ADD_USER

Add a new user, or change the password for an existing user.

AP_ADD_PROFILES

Add to the profiles for an existing user.

user_id

User identifier. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

Some CPI-C implementations have a maximum user ID length of 8 characters. If you specify a user ID of 9 or 10 characters, CPI-C applications running on other systems may not be able to access applications on the SNAplus2 system using this user ID and password.

password_chars.description

A null-terminated text string (0-31 characters followed by a null character) describing the user ID and password. This string is for information only; it is stored in the node's configuration file and returned on the QUERY_USERID_PASSWORD verb, but SNAplus2 does not make any other use of it.

password_chars.profile_count

Number of profiles. This parameter is normally set to zero; see *password_chars.profiles* below for more information.

password_chars.password

User's password. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

Some CPI-C implementations have a maximum password length of 8 characters. If you specify a password of 9 or 10 characters, CPI-C applications running on other systems may not be able to access applications on the SNAplus2 system using this user

ID and password.

Whatever value the application supplies for this parameter is immediately replaced by the encrypted version of the password. Therefore, the value supplied for the *password_chars.password* parameter is never written out.

password_chars.profiles

Profile names associated with the user ID and password. Each of these is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

If a remote TP uses this user ID and password to contact the local TP, and specifies a profile on the Attach, this must match one of the profile names defined here. Check with the remote System Administrator to determine if a profile will be used; for each profile that will be used, specify the profile name as one of the *profiles* parameters on this verb. In most cases, profile names are not used, and so there is no need to specify them on this verb; set *password_chars.profile_count* to zero and do not specify any profiles.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PASSWORD

The *password* parameter contained a character that

was not valid.

AP_INVALID_PROFILE

One or more of the specified profiles was not valid.

AP_INVALID_UPDATE_TYPE

The *define_type* parameter was not set to a valid value.

AP_INVALID_USERID

The *user_id* parameter contained a character that was not valid.

AP_NO_PROFILES

The verb was used to add profiles to an existing user, but no profiles were specified.

AP_TOO_MANY_PROFILES

The *profile_count* parameter was not set to a valid value.

AP_UNKNOWN_USER

The verb was used to add profiles to an existing user, but the *user_id* parameter did not match an existing user ID.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_ADJACENT_LEN_NODE

DELETE_ADJACENT_LEN_NODE removes entries in the node directory database for an adjacent LEN node and its associated LUs, or removes LU entries for the LEN node without removing the LEN node itself. It is equivalent to issuing a series of DELETE_DIRECTORY_ENTRY verbs for the LEN node and its associated LUs.

VCB Structure

```
typedef struct delete_adjacent_len_node
{
     AP_UINT16         opcode;              /* verb operation code          */
     unsigned char     reserv2;             /* reserved                     */
     unsigned char     format;              /* reserved                     */
     AP_UINT16         primary_rc;          /* primary return code          */
     AP_UINT32         secondary_rc;        /* secondary return code        */
     unsigned char     cp_name[17];         /* CP name                      */
     unsigned char     num_of_lus;          /* number of LUs                */
     unsigned char     lu_names[10][8];     /* LU names                     */
     unsigned char     wildcard_lus;        /* wildcard LUs                 */
} DELETE_ADJACENT_LEN_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_ADJACENT_LEN_NODE
cp_name	The fully qualified name of the CP in the adjacent LEN node. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.
num_of_lus	The number of LUs to be deleted, in the range 1 to 10. To delete the entire LEN node definition, specify zero.
lu_names	The names of the LUs on the LEN node to be deleted. Each name is an 8-byte type-A EBCDIC character

string, right-padded with EBCDIC spaces. Do not specify any LU names if you are deleting the entire LEN node definition (if *num_of_lus* is zero).

You can specify a "wildcard" LU name to match multiple LU names, by specifying only the initial characters of the name. For example, the wildcard LU name APPN.LU will match APPN.LUNAME or APPN.LU01 (but will not match APPN.NAMELU). However, all the LU names specified on a single verb must be of the same type (wildcard or explicit), as defined by the *wildcard_lus* parameter below. To remove both types of LU names from the same LEN node, use multiple DELETE_ADJACENT_LEN_NODE verbs.

wildcard_lus Indicates whether the specified LU names are wildcard entries or explicit LU names. Possible values are:

AP_YES

The specified LU names are wildcard entries.

AP_NO

The specified LU names are explicit entries.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CP_NAME

The *cp_name* parameter contained a character that was not valid.

AP_INVALID_LU_NAME

One or more of the specified LU names contained a character that was not valid.

AP_INVALID_NUM_LUS

The *num_of_lus* parameter was not in the valid range.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_INVALID_CP_NAME

The specified CP name does not exist.

AP_INVALID_LU_NAME

One or more of the specified LU names does not exist.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_BACKUP

An application uses this verb to delete a server from the list of backup master servers in the sna.net file, so that this server can no longer act as the master configuration file server.

You can use this verb to delete any server in the list, including the master server, whether or not the SNA software is running on the server you are deleting. The only restriction is that the list must always contain at least one server on which the SNA software is running (so that this server can take over as the master server); you cannot delete a server if it is the only server in the list or if it is the only server listed on which the SNA software is running.

This verb must be issued to the sna.net file.

VCB Structure

```
typedef struct delete_backup
{
     AP_UINT16          opcode;              /* verb operation code         */
     unsigned char      reserv2;             /* reserved                    */
     unsigned char      format;              /* reserved                    */
     AP_UINT16          primary_rc;          /* primary return code         */
     AP_UINT32          secondary_rc;        /* secondary return code       */
     unsigned char      backup_name[64];     /* name of server to delete    */
     unsigned char      reserv4[4];          /* reserved                    */
} DELETE_BACKUP;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_BACKUP
backup_name	The name of the server being deleted from the list of backup servers.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_RECORD_NOT_FOUND

The server name specified is not listed in the file.

AP_CANT_DELETE_LAST_BACKUP

The server name cannot be deleted from the list, because it is the only server listed on which the SNA software is running (and hence the only server that can currently act as the master). Before attempting to delete it, either start the SNA software on one or more of the other servers listed, or add one or more new backup servers (using ADD_BACKUP) and ensure that the SNA software is started on these servers.

AP_INVALID_TARGET

The target handle on the NOF API call specified a configuration file or a node. This verb must be issued to the sna.net file.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_CN

DELETE_CN deletes a connection network, or deletes selected ports from a connection network.

This verb is valid only at an end node, and not at a LEN node.

VCB Structure

```
typedef struct delete_cn
{
     AP_UINT16         opcode;             /* verb operation code           */
     unsigned char     reserv2;            /* reserved                      */
     unsigned char     format;             /* reserved                      */
     AP_UINT16         primary_rc;         /* primary return code           */
     AP_UINT32         secondary_rc;       /* secondary return code         */
     unsigned char     fqcn_name[17];      /* name of Connection Network    */
     unsigned char     reserv1;            /* reserved                      */
     AP_UINT16         num_ports;          /* number of ports to delete     */
     unsigned char     port_name[8][8];    /* names of ports to delete      */
} DELETE_CN;
```

Supplied Parameters

The application supplies the following parameters:

opcode — AP_DELETE_CN

fqcn_name — Fully qualified name of the connection network. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

num_ports — Specify zero to delete the connection network, or the number of ports to be deleted if you are removing ports instead of deleting the connection network.

port_name — If you are removing ports (if *num_ports* is nonzero), specify the names of the ports to be deleted. Each port name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. If you are deleting the connection network (if *num_ports* is

zero), these names must be set to binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_CN_NAME
	AP_INVALID_NUM_PORTS_SPECIFIED

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_COS

DELETE_COS deletes a class of service entry. Only locally defined classes of service can be deleted; the default classes of service defined by SNA cannot be deleted.

If the node supports mode to COS mapping (as defined by the *mode_to_cos_map_supp* parameter on DEFINE_NODE) and the configuration includes modes that are mapped to the COS that you are deleting, SNAplus2 will remap these modes to the default COS (specified by a DEFINE_MODE verb with a null mode name) or to the SNA-defined COS #CONNECT if no default COS is specified.

VCB Structure

```
typedef struct delete_cos
{
     AP_UINT16          opcode;             /* verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* primary return code          */
     AP_UINT32          secondary_rc;       /* secondary return code        */
     unsigned char      cos_name[8];        /* class of service name        */
} DELETE_COS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_COS

cos_name Class of service name. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_COS_NAME_NOT_DEFD

The supplied name is not the name of a COS defined on the SNAplus2 system.

AP_SNA_DEFD_COS_CANT_BE_DELETE

The supplied name is the name of one of the SNA-defined classes of service, which cannot be deleted.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_CPIC_SIDE_INFO

This verb deletes an entry from the side information table.

Note the difference between this verb and the CPI-C function `Delete_CPIC_Side_Information()`. This verb modifies a configuration file, so that it affects all SNAplus2 CPI-C applications. The CPI-C function modifies the application's own copy in memory of the side information table, and does not affect any other CPI-C applications.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct delete_cpic_side_info
{
     AP_UINT16          opcode;             /* verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* primary return code          */
     AP_UINT32          secondary_rc;       /* secondary return code        */
     unsigned char      reserv2a[8];        /* reserved                     */
     unsigned char      sym_dest_name[8];   /* Symbolic destination name    */
} DELETE_CPIC_SIDE_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_CPIC_SIDE_INFO

sym_dest_name

Symbolic destination name which identifies the side information entry. This is an 8-byte ASCII string, consisting of uppercase A-Z and digits 0-9, padded on the right with spaces if necessary.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc AP_OK

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_SYM_DEST_NAME

The *sym_dest_name* parameter was not the name of a defined CPI-C side information entry.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_DIRECTORY_ENTRY

DELETE_DIRECTORY_ENTRY deletes an entry in the Network Directory.

If the entry for a parent resource is deleted, then all entries for child resources associated with it are also deleted. For example, if you delete the entry for a network node that is the parent of an end node, then the entries for the end node and all LUs associated with both nodes (including wildcard LU entries) are deleted as well as the entry for the network node.

VCB Structure

```
typedef struct delete_directory_entry
{
   AP_UINT16          opcode;             /* verb operation code            */
   unsigned char      reserv2;            /* reserved                       */
   unsigned char      format;             /* reserved                       */
   AP_UINT16          primary_rc;         /* primary return code            */
   AP_UINT32          secondary_rc;       /* secondary return code          */
   unsigned char      resource_name[17];  /* fully qualified resource name  */
   unsigned char      reserv3;            /* reserved                       */
   AP_UINT16          resource_type;      /* resource type                  */
} DELETE_DIRECTORY_ENTRY;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_DIRECTORY_ENTRY

resource_name Fully qualified name of the resource to be deleted. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

resource_type Specifies the type of the resource to be deleted. Possible values are:

AP_ENCP_RESOURCE

End node or LEN node

AP_NNCP_RESOURCE

Network node

AP_LU_RESOURCE

LU

AP_WILDCARD_LU_RESOURCE

Wildcard LU name.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_FQ_LU_NAME

The *resource_name* parameter was not the name of a defined directory entry.

AP_INVALID_RESOURCE_TYPE

The *resource_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_DLC

DELETE_DLC deletes a DLC. This verb also deletes the following:

- All ports, link stations and connection network TGs associated with the DLC
- All PUs associated with LSs on the DLC, all LUs owned by these PUs, and all LU-LU passwords associated with these LUs.

VCB Structure

```
typedef struct delete_dlc
{
     AP_UINT16           opcode;             /* verb operation code         */
     unsigned char       reserv2;            /* reserved                    */
     unsigned char       format;             /* reserved                    */
     AP_UINT16           primary_rc;         /* primary return code         */
     AP_UINT32           secondary_rc;       /* secondary return code       */
     unsigned char       dlc_name[8];        /* name of DLC                 */
} DELETE_DLC;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_DLC

dlc_name Name of DLC to be deleted. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_DLC_NAME

The supplied DLC name was not the name of a DLC defined on the SNAplus2 system.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_DLC_ACTIVE

The DLC cannot be deleted because it is currently active. Use the STOP_DLC verb to stop it before attempting to delete it.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_DOWNSTREAM_LU

This verb is used to delete a downstream LU.

VCB Structure

```
typedef struct delete_downstream_lu
{
     AP_UINT16          opcode;               /* verb operation code        */
     unsigned char      reserv2;              /* reserved                   */
     unsigned char      format;               /* reserved                   */
     AP_UINT16          primary_rc;           /* primary return code        */
     AP_UINT32          secondary_rc;         /* secondary return code      */
     unsigned char      dslu_name[8];         /* Downstream LU name         */
} DELETE_DOWNSTREAM_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_DOWNSTREAM_LU

dslu_name Name of the downstream LU that is being deleted. This is an 8-byte type A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LU_NAME

The *dslu_name* parameter contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_LU_NAME

The *dslu_name* parameter did not match any defined downstream LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration; this is defined by the *pu_conc_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_DOWNSTREAM_LU_RANGE

This verb is used to delete a range of downstream LUs.

The supplied parameters to this verb include a base name for the LUs and the range of NAU addresses. The LU names to be deleted are determined by combining the base name with the NAU addresses. For example, a base name of LUNME combined with a NAU range of 11 to 14 would delete the LUs LUNME011, LUNME012, LUNME013, and LUNME014.

All LUs with names in the specified range are deleted; SNAplus2 does not return an error if one or more names in the range do not exist.

VCB Structure

```
typedef struct delete_downstream_lu_range
{
    AP_UINT16         opcode;              /* verb operation code            */
    unsigned char     reserv2;             /* reserved                       */
    unsigned char     format;              /* reserved                       */
    AP_UINT16         primary_rc;          /* primary return code            */
    AP_UINT32         secondary_rc;        /* secondary return code          */
    unsigned char     dslu_base_name[5];   /* LU base name                   */
    unsigned char     min_nau;             /* Minimum NAU address in range   */
    unsigned char     max_nau;             /* Maximum NAU address in range   */
} DELETE_DOWNSTREAM_LU_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_DOWNSTREAM_LU_RANGE

dslu_base_name

Base name for the names of the LUs. This is a 5-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the base name is less than 5 characters. SNAplus2 determines the names of the LUs to be deleted by appending the 3-digit decimal value of each NAU address to this

name.

min_nau

NAU address of the first LU, in the range 1-255.

max_nau

NAU address of the last LU, in the range 1-255.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_NAU_ADDRESS

The *min_nau* or *max_nau* parameter was not valid.

AP_INVALID_LU_NAME

The *dslu_base_name* parameter contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_LU_NAME

There were no LUs defined with names in the specified

range.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration; this is defined by the *pu_conc_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_DSPU_TEMPLATE

The DELETE_DSPU_TEMPLATE verb deletes a specific downstream physical unit (DSPU) template that was previously defined using a DEFINE_DSPU_TEMPLATE verb.

VCB Structure

```
typedef struct delete_dspu_template
{
    AP_UINT16         opcode;               /* verb operation code           */
    unsigned char     reserv2;              /* reserved                      */
    unsigned char     format;               /* reserved                      */
    AP_UINT16         primary_rc;           /* primary return code           */
    AP_UINT32         secondary_rc;         /* secondary return code         */
    unsigned char     template_name;        /* name of template              */
    AP_UINT16         num_of_dslu_templates /* number of dslu templates      */
```

Supplied Parameters

Supplied parameters are:

opcode

AP_DELETE_DSPU_TEMPLATE

template_name

Name of the DSPU template to be deleted. Specify 1-8 locally displayable characters.

num_of_dslu_templates

Number of DSLU templates to be deleted. Specify a value in the range 1-255, or specify 0 (zero) to delete the entire DSPU template.

The subrecord dslu_template contains the following parameters:

min_nau

Minimum NAU address in the range of DSLU templates to be deleted. Specify a value in the range 1-255.

max_nau

Maximum NAU address in the range of DSLU templates to be deleted. Specify a value in the range 1-255.

allow_timeout

Specifies whether SNAplus2 is allowed to timeout host LUs used by this downstream LU if the session is left inactive for the timeout period specified on the host LU definition. Possible values are:

AP_YES

SNAplus2 is allowed to timeout host LUs used by this downstream LU.

AP_NO

SNAplus2 is not allowed to timeout host LUs used by this downstream LU.

delayed_logon

Specifies whether SNAplus2 delays connecting the downstream LU to the host LU until the first data is received from the downstream LU. Instead, a simulated logon screen is sent to the downstream LU. Possible values are:

AP_YES

SNAplus2 delays connecting the downstream LU to the host LU until the first data is received from the downstream LU.

AP_NO

SNAplus2 does not delay connecting the downstream LU to the host LU until the first data is received from the downstream LU.

host_lu

Name of the host LU or host LU pool onto which all the downstream LUs within the range will be mapped.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_TEMPLATE_NAME

The template specified by the *template_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_EMULATOR_USER

DELETE_EMULATOR_USER is used to do one of the following:

- Delete a user of the 3270 emulation program or 5250 emulation program, so that this user can no longer use the program.
- Delete one or more of the user's sessions but leave the user configured.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct delete_emulator_user
{
   AP_UINT16        opcode;               /* verb operation code           */
   unsigned char    reserv2;              /* reserved                      */
   unsigned char    format;               /* reserved                      */
   AP_UINT16        primary_rc;           /* primary return code           */
   AP_UINT32        secondary_rc;         /* secondary return code         */
   unsigned char    user_name[32];        /* user name                     */
   AP_UINT32        num_sessions;         /* number of sessions to delete  */
   unsigned char    delete_options;       /* delete all sessions / delete  */
                                          /*  user?                        */
   unsigned char    session_names[10][8]; /* names of sessions to be       */
                                          /*  deleted                      */
} DELETE_EMULATOR_USER;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_EMULATOR_USER

user_name The name of the user. This is an ASCII string of 1-32 characters, padded on the right with spaces if the name is shorter than 32 characters, which must match a previously-defined emulator user name.

num_sessions The number of sessions to be deleted, as follows:

- To delete one or more of the user's sessions but

leave other sessions configured, specify the number of sessions that are being deleted. Each of these must be defined by its session name, as described below.

- To delete all sessions, or to delete the user, specify zero in this parameter and do not include any session names. Specify the type of deletion required in the *delete_options* parameter below.

delete_options If the *num_sessions* parameter (see above) is nonzero, this parameter is ignored. If *num_sessions* is zero, specify one of the following values:

AP_3270_SESSION

Delete all 3270 sessions but leave any 5250 sessions unchanged.

AP_5250_SESSION

Delete all 5250 sessions but leave any 3270 sessions unchanged.

AP_ALL_SESSIONS

Delete all sessions of both types but leave the user configured.

AP_DELETE_USER

Delete the user and all the user's sessions.

For each session to be deleted, up to the number specified in *num_sessions*, the following parameter is required:

delete_session_data.session_name

The long name of the session. This is an ASCII string of 1-8 characters, which must match a session name defined for this user. This parameter is ignored unless the *delete_options* parameter is set to AP_DELETE_USER.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_EMULATOR_USER

The specified user name was not defined as an emulation program user name.

AP_INVALID_USER_SESSION

The specified session name did not match the name of a session defined for this user.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_FOCAL_POINT

The DELETE_FOCAL_POINT verb removes the definition of a focal point for a specified MS category (either the main focal point for that category or a backup focal point). If the defined focal point application is active and acting as the current focal point for that category, SNAplus2 sends an MS_CAPABILITIES message to the focal point to revoke it so that it no longer acts as the focal point.

VCB Structure

```
typedef struct delete_focal_point
{
     AP_UINT16         opcode;           /* verb operation code            */
     unsigned char     reserv2;          /* reserved                       */
     unsigned char     format;           /* reserved                       */
     AP_UINT16         primary_rc;       /* primary return code            */
     AP_UINT32         secondary_rc;     /* secondary return code          */
     unsigned char     reserved;         /* reserved                       */
     unsigned char     ms_category[8];   /* management services category   */
     unsigned char     type;             /* type of focal point            */
} DELETE_FOCAL_POINT;
```

Supplied Parameters

The application supplies the following parameters:

opcode — AP_DELETE_FOCAL_POINT

ms_category — Management Services category. This may be either one of the category names specified in the MS Discipline-Specific Application Programs table of *Systems Network Architecture: Management Services Reference* (see "Related Publications"), padded with EBCDIC spaces (0x40), or a user-defined category. A user-defined category name is an 8-byte type-1134 EBCDIC string, padded with EBCDIC spaces (0x40) if necessary.

type — Specifies the type of the focal point that is being deleted. Possible values are:

AP_ACTIVE

The currently active focal point (which may be of any type) is revoked.

AP_IMPLICIT

The implicit definition (defined using DEFINE_FOCAL_POINT with backup set to AP_NO) is removed. If this focal point is currently active, then it is revoked.

AP_BACKUP

The backup definition (defined using DEFINE_FOCAL_POINT with backup set to AP_YES) is removed. If this focal point is currently active, then it is revoked.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CATEGORY_NAME

The supplied category name contained a character that was not valid.

AP_INVALID_TYPE

The *type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_INTERNAL_PU

DELETE_INTERNAL_PU deletes a DLUR-served local PU. The PU can be deleted only if it does not have an active SSCP-PU session.

VCB Structure

```
typedef struct delete_internal_pu
{
     AP_UINT16          opcode;          /* verb operation code            */
     unsigned char      reserv2;         /* reserved                       */
     unsigned char      format;          /* reserved                       */
     AP_UINT16          primary_rc;      /* primary return code            */
     AP_UINT32          secondary_rc;    /* secondary return code          */
     unsigned char      pu_name[8];      /* internal PU name               */
} DELETE_INTERNAL_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_INTERNAL_PU

pu_name Name of the internal PU that is being deleted. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_PU_NAME

The *pu_name* parameter was not the name of a defined internal PU.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_PU_NOT_RESET

The PU cannot be deleted because it still has an active PU-SSCP session.

AP_INVALID_PU_TYPE

The specified PU is a remote PU and not an internal PU.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The node does not support DLUR; this is defined by the *dlur_supported* parameter on DEFINE_NODE.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LOCAL_LU

The DELETE_LOCAL_LU verb deletes a local LU, and also deletes any LU-LU passwords associated with the local LU.

VCB Structure

```
typedef struct delete_local_lu
{
     AP_UINT16           opcode;             /* verb operation code         */
     unsigned char       reserv2;            /* reserved                    */
     unsigned char       format;             /* reserved                    */
     AP_UINT16           primary_rc;         /* primary return code         */
     AP_UINT32           secondary_rc;       /* secondary return code       */
     unsigned char       lu_name[8];         /* local LU name               */
} DELETE_LOCAL_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_LOCAL_LU
lu_name	Name of the local LU to be deleted. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_CANT_DELETE_CP_LU

The supplied LU name was blank (indicating the LU associated with the CP); this LU cannot be deleted.

AP_INVALID_LU_NAME

The supplied LU name is not the name of a local LU defined on the SNAplus2 system.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LS

DELETE_LS deletes a defined Link Station (LS). This verb also deletes the PU associated with the LS, all LUs owned by this PU, and all LU-LU passwords associated with these LUs. The LS cannot be deleted if it is active.

VCB Structure

```
typedef struct delete_ls
{
     AP_UINT16          opcode;           /* verb operation code          */
     unsigned char      reserv2;          /* reserved                     */
     unsigned char      format;           /* reserved                     */
     AP_UINT16          primary_rc;       /* primary return code          */
     AP_UINT32          secondary_rc;     /* secondary return code        */
     unsigned char      ls_name[8];       /* name of link station         */
} DELETE_LS;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_LS
ls_name	Name of link station being deleted. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LINK_NAME

The supplied LS name contains a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_LS_ACTIVE

The LS cannot be deleted because it is currently active.

AP_INVALID_LINK_NAME

The supplied LS name is not the name of an LS defined on the SNAplus2 system.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LU62_TIMEOUT

The DELETE_LU62_TIMEOUT verb deletes a definition of an LU type 6.2 session timeout that was defined previously with a DEFINE_LU62_TIMEOUT verb.

VCB Structure

```
typedef struct delete_lu62_timeout
{
    AP_UINT16         opcode;                /* verb operation code        */
    unsigned char     reserv2;
    unsigned char     format;                /* reserved                   */
    AP_UINT16         primary_rc;            /* primary return code        */
    AP_UINT32         secondary_rc;          /* secondary return code      */
    unsigned char     resource_type;         /* resource type              */
    unsigned char     resource_name[17];     /* resource name              */
} DELETE_LU62_TIMEOUT;
```

Supplied Parameters

Supplied parameters are:

opcode AP_DELETE_LU62_TIMEOUT

resource_type Specifies the type of timeout being deleted. Possible values are:

AP_GLOBAL_TIMEOUT

Delete timeouts that apply to all LU 6.2 sessions for the local node.

AP_LOCAL_LU_TIMEOUT

Delete timeouts that apply to all LU 6.2 sessions for the local LU specified in the *resource_name* parameter.

AP_PARTNER_LU_TIMEOUT

Delete timeouts that apply to all LU 6.2 sessions to the partner LU specified in the *resource_name* parameter.

AP_MODE_TIMEOUT

Delete timeouts that apply to all LU 6.2 sessions on the mode specified in the *resource_name* parameter.

resource_name Name of the resource whose timeout is being deleted. This value can be one of the following:

- If *resource_type* is set to AP_GLOBAL_TIMEOUT, do not specify this parameter.
- If *resource_type* is set to AP_LOCAL_LU_TIMEOUT, specify 1-8 locally displayable type-A characters as a local LU name.
- If *resource_type* is set to AP_PARTNER_LU_TIMEOUT, specify the fully qualified name of the partner LU as follows: 17 locally displayable type-A characters consisting of a 1-8 character network name, followed by a period, followed by a 1-8 character partner LU name.
- If *resource_type* is set to AP_MODE_TIMEOUT, specify 1-8 locally displayable type-A characters as a mode name.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_RESOURCE_TYPE

The value specified in the *resource_type* parameter was not valid.

AP_INVALID_LU_NAME

The LU name specified in the *resource_name* parameter was not valid.

AP_INVALID_PARTNER_LU

The partner LU name specified in the *resource_name* parameter was not valid.

AP_INVALID_MODE_NAME

The mode name specified in the *resource_name* parameter was not valid.

AP_GLOBAL_TIMEOUT_NOT_DEFINED

The value AP_GLOBAL_TIMEOUT was specified for the *resource_type* parameter but there is no defined global timeout.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LU_0_TO_3

This verb is used to delete an LU used for 3270 emulation or LUA (an LU of type 0-3).

VCB Structure

```
typedef struct delete_lu_0_to_3
{
     AP_UINT16           opcode;          /* verb operation code          */
     unsigned char       reserv2;         /* reserved                     */
     unsigned char       format;          /* reserved                     */
     AP_UINT16           primary_rc;      /* primary return code          */
     AP_UINT32           secondary_rc;    /* secondary return code        */
     unsigned char       lu_name[8];      /* LU name                      */
} DELETE_LU_0_TO_3;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_LU_0_TO_3
lu_name	Name of the local LU to be deleted. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LU_NAME

The supplied LU name contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_LU_NAME

The supplied LU name is not the name of an LU defined on the SNAplus2 system.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LU_0_TO_3_RANGE

This verb is used to delete a range of LUs used for 3270 emulation or LUA (type 0-3 LUs).

The supplied parameters to this verb include a base name for the LUs and the range of NAU addresses. The LU names to be deleted are determined by combining the base name with the NAU addresses. For example, a base name of `LUNME` combined with a NAU range of `11-14` would delete the LUs `LUNME011`, `LUNME012`, `LUNME013`, and `LUNME014`.

All LUs with names in the specified range are deleted; SNAplus2 does not return an error if one or more names in the range do not exist.

VCB Structure

```
typedef struct delete_lu_0_to_3_range
{
    AP_UINT16         opcode;           /* verb operation code             */
    unsigned char     reserv2;          /* reserved                        */
    unsigned char     format;           /* reserved                        */
    AP_UINT16         primary_rc;       /* primary return code             */
    AP_UINT32         secondary_rc;     /* secondary return code           */
    unsigned char     base_name[6];     /* Base name                       */
    unsigned char     min_nau;          /* Minimum NAU address in range    */
    unsigned char     max_nau;          /* Maximum NAU address in range    */
    unsigned char     name_attributes;  /* Extension type                  */
    unsigned char     base_number;      /* First extension number          */
    unsigned char     reserv5[16];      /* reserved                        */
} DELETE_LU_0_TO_3_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

`AP_DELETE_LU_0_TO_3_RANGE`

base_name

Base name for the names of the LUs. This is a 5-byte type-A EBCDIC string (starting with a letter), padded

on the right with EBCDIC spaces if the base name is less than 5 characters. SNAplus2 determines the names of the LUs to be deleted by appending the 3-digit decimal value of each NAU address to this name.

min_nau

NAU address of the first LU, in the range 1-255.

max_nau

NAU address of the last LU, in the range 1-255.

name_attributes

Specifies the extension type of the LUs. Possible values are:

AP_NONE

LU names have numbers that correspond to the NAU numbers. The numbers are specified in decimal and the *base_name* parameter can contain only five characters.

AP_USE_BASE_NUMBER

Start deleting the LUs in the range from the value specified in the *base_number* parameter.

AP_USE_HEX_IN_NAME

The extension to the LU name is in hex rather than decimal. The *base_name* parameter can contain 6 characters if this value is specified.

base_number

If AP_USE_BASE_NUMBER is specified in the *name_attributes* parameter, specify a number from which to start deleting the LUs in the range. This value will be used instead of the value of the *min_nau* parameter.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_NAU_ADDRESS

The *min_nau* or *max_nau* parameter was not valid.

AP_INVALID_LU_NAME

The *base_name* parameter contained a character that was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_LU_NAME

There were no LUs defined with names in the specified range.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LU_LU_PASSWORD

DELETE_LU_LU_PASSWORD deletes an LU-LU password associated with a local LU. LU-LU passwords are deleted automatically when the local LU is deleted; you need only use this verb if you need to remove the password but leave the LU configured.

VCB Structure

```
typedef struct delete_lu_lu_password
{
    AP_UINT16        opcode;            /* verb operation code             */
    unsigned char    reserv2;           /* reserved                        */
    unsigned char    format;            /* reserved                        */
    AP_UINT16        primary_rc;        /* primary return code             */
    AP_UINT32        secondary_rc;      /* secondary return code           */
    unsigned char    lu_name[8];        /* LU name                         */
    unsigned char    lu_alias[8];       /* local LU alias                  */
    unsigned char    fqplu_name[17];    /* fully qualified partner LU name */
    unsigned char    reserv3;           /* reserved                        */
} DELETE_LU_LU_PASSWORD;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_LU_LU_PASSWORD

lu_name LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

fqplu_name Fully qualified LU name for the partner LU, as defined to SNAplus2. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_PLU_NAME

The *fqplu_name* parameter was not valid.

AP_INVALID_LU_NAME

The *lu_name* parameter was not valid.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_LU_POOL

DELETE_LU_POOL is used to do one of the following:

- Remove one or more LUs from a pool
- Remove all LUs from a pool and delete the pool

This verb does not delete the LUs; they remain defined, but are not associated with any pool.

VCB Structure

```
typedef struct delete_lu_pool
{
     AP_UINT16          opcode;             /* verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* primary return code          */
     AP_UINT32          secondary_rc;       /* secondary return code        */
     unsigned char      pool_name[8];       /* LU pool name                 */
     AP_UINT16          num_lus;            /* Number of specified LUs      */
     unsigned char      lu_names[10][8];    /* LU names                     */
} DELETE_LU_POOL;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_LU_POOL
pool_name	Name of the LU pool. This is an 8-byte EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.
num_lus	The number of LUs to be removed (the number of LU names in the *lu_names* list). The range is 1-10 when removing LUs from a pool without deleting it. To remove all LUs from the pool and delete the pool, specify zero.
lu_names	To remove one or more LUs from the pool without deleting the pool, specify the names of the LUs to be

removed. The number of names specified must match the *num_lus* parameter. Each name is an 8-byte type A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

If *num_lus* is set to zero, to remove all LUs from the pool and delete the pool, this parameter is not used.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_POOL_NAME

The supplied pool name was not valid.

AP_INVALID_LU_NAME

One or more of the specified LU names did not match the name of an LU in the pool.

AP_INVALID_NUM_LUS

The supplied *num_lus* parameter was not in the valid range.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_MODE

DELETE_MODE deletes the definition of a mode. You cannot delete SNA-defined modes such as SNASVCMG and CPSVCMG.

VCB Structure

```
typedef struct delete_mode
{
     AP_UINT16           opcode;           /* verb operation code          */
     unsigned char       reserv2;          /* reserved                     */
     unsigned char       format;           /* reserved                     */
     AP_UINT16           primary_rc;       /* primary return code          */
     AP_UINT32           secondary_rc;     /* secondary return code        */
     unsigned char       mode_name[8];     /* mode name                    */
} DELETE_MODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_MODE

mode_name Name of the mode. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_CP_OR_SNA_SVCMG_UNDELETABLE

The specified mode name is one of the SNA-defined mode names, and cannot be deleted.

AP_MODE_NAME_NOT_DEFD

The specified mode name is not the name of a mode defined on the SNAplus2 system.

AP_DEL_MODE_DEFAULT_SPCD

The specified mode was defined as the default mode using the DEFINE_DEFAULTS verb, so it cannot be deleted.

AP_MODE_UNDELETABLE

The specified mode name is one of the SNA-defined mode names, and cannot be deleted.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_PARTNER_LU

The DELETE_PARTNER_LU verb deletes a partner LU definition.

VCB Structure

```
typedef struct delete_partner_lu
{
    AP_UINT16         opcode;           /* verb operation code             */
    unsigned char     reserv2;          /* reserved                        */
    unsigned char     format;           /* reserved                        */
    AP_UINT16         primary_rc;       /* primary return code             */
    AP_UINT32         secondary_rc;     /* secondary return code           */
    unsigned char     fqplu_name[17];   /* fully qualified partner LU name  */
} DELETE_PARTNER_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_PARTNER_LU

fqplu_name Fully qualified LU name for the partner LU to be deleted. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_PLU_NAME The supplied *fqplu_name* parameter did not match any defined partner LU name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_PORT

DELETE_PORT deletes a port. This verb also deletes the following:

- All link stations and connection network TGs associated with the port.
- All PUs associated with LSs on the port, all LUs owned by these PUs, and all LU-LU passwords associated with these LUs.

The port must be inactive when the verb is issued.

VCB Structure

```
typedef struct delete_port
{
     AP_UINT16           opcode;           /* verb operation code          */
     unsigned char       reserv2;          /* reserved                     */
     unsigned char       format;           /* reserved                     */
     AP_UINT16           primary_rc;       /* primary return code          */
     AP_UINT32           secondary_rc;     /* secondary return code        */
     unsigned char       port_name[8];     /* name of port                 */
} DELETE_PORT;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_PORT

port_name Name of port being deleted. This is an 8-byte ASCII string, right-paddded with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_PORT_NAME

The specified port name was not the name of a port defined on the SNAplus2 system.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_PORT_ACTIVE

The specified port cannot be deleted because it is currently active.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_RCF_ACCESS

DELETE_RCF_ACCESS prevents access to the SNAplus2 Remote Command Facility (RCF), which was previously specified using DEFINE_RCF_ACCESS. For more information about RCF, see the *HP-UX SNAplus2 Administration Guide*.

This verb prevents access to both SPCF and UCF. To allow access to one of them but prevent access to the other, use DEFINE_RCF_ACCESS.

This verb must be issued to the domain configuration file. SNAplus2 acts on the RCF access parameters during node startup; if RCF access is deleted while a node is running, the change does not take effect on the server where the node is running until the node is stopped and restarted.

VCB Structure

```
typedef struct delete_rcf_access
{
     AP_UINT16           opcode;           /* Verb operation code          */
     unsigned char       reserv2;          /* reserved                     */
     unsigned char       format;           /* reserved                     */
     AP_UINT16           primary_rc;       /* primary return code          */
     AP_UINT32           secondary_rc;     /* secondary return code        */
} DELETE_RCF_ACCESS;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_DELETE_RCF_ACCESS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_RJE_WKSTN

DELETE_RJE_WKSTN deletes an RJE workstation.

Before deleting the workstation, check with the users of the workstation to ensure that it is not being used. If you delete a workstation while it is running, the only RJE commands that will subsequently be accepted for that workstation are the commands to stop the workstation and to clear out its directory structure. For more information about these commands, see the *HP-UX SNAplus2 RJE Users Guide*.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct delete_rje_wkstn
{
     AP_UINT16          opcode;                /* verb operation code       */
     unsigned char      reserv2;               /* reserved                  */
     unsigned char      format;                /* reserved                  */
     AP_UINT16          primary_rc;            /* primary return code       */
     AP_UINT32          secondary_rc;          /* secondary return code     */
     unsigned char      workstation_name[4];   /* workstation name          */
} DELETE_RJE_WKSTN;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_RJE_WKSTN

workstation_name

The name of the RJE workstation to be deleted.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_WORKSTATION_NAME The specified *workstation_name* parameter did not match any RJE workstation name.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_SECURITY_ACCESS_LIST

DELETE_SECURITY_ACCESS_LIST is used to do one of the following:

- Delete a security access list.
- Delete one or more users from a security access list but leave the list configured.

You can delete a user name from the security access list regardless of whether there are active conversations that were set up using that user name. Deleting the user name does not affect the active conversations, but the invoking program will not be able to set up any further conversations using the deleted user name.

VCB Structure

The DELETE_SECURITY_ACCESS_LIST verb contains a variable number of `security_user_name` structures. These define the user names to be deleted from the security access list. The user name structures are included at the end of the `delete_security_access_list` structure. The number of these structures is specified by the *`num_users`* parameter.

```
typedef struct delete_security_access_list
{
   AP_UINT16         opcode;        /* verb operation code          */
   unsigned char     reserv2;       /* reserved                     */
   unsigned char     format;        /* reserved                     */
   AP_UINT16         primary_rc;    /* primary return code          */
   AP_UINT32         secondary_rc;  /* secondary return code        */
   unsigned char     list_name[14]; /* name of this list            */
   unsigned char     reserv3[2];    /* reserved                     */
   AP_UINT32         num_users;     /* number of users to delete    */
} DELETE_SECURITY_ACCESS_LIST;


typedef struct security_user_name
{
unsigned char     user_name[10];    /* user name to delete          */
} SECURITY_USER_NAME;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_SECURITY_ACCESS_LIST

list_name The name of the security access list being deleted, or the list from which user names are being deleted. This is an ASCII string of 1 - 14 characters, padded on the right with spaces if the name is shorter than 14 characters, which must match a previously-defined security access list name.

num_users The number of user names to be deleted from the security access list, as follows

- To delete one or more user names from the list but leave other user names configured, specify the number of user names that are being deleted. Each of these must be defined by a user name structure, as described below.
- To delete the entire security access list, specify zero in this parameter and do not include any user names

For each user name to be delete, up to the number specified in *num_users*, the following parameter is required:

user_name The user name being deleted. This must match a user name that is currently defined for this security access list.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LIST_NAME

The specified security access list name was to defined as a security access list name.

AP_INVALID_USER_NAME

One or more of the specified user names did not match the name of a user defined for this security access list.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_TN3270_ACCESS

DELETE_TN3270_ACCESS is used to do one of the following:

- Delete a TN3270 user, so that this user can no longer use TN server to access a host.
- Delete one or more of the user's sessions but leave the user configured.

VCB Structure

```
typedef struct delete_tn3270_access
{
   AP_UINT16        opcode;              /* verb operation code            */
   unsigned char    reserv2;             /* reserved                       */
   unsigned char    format;              /* reserved                       */
   AP_UINT16        primary_rc;          /* primary return code            */
   AP_UINT32        secondary_rc;        /* secondary return code          */
   AP_UINT16        default_record;      /* is this the DEFAULT record?    */
   unsigned char    client_address[68];  /* address of TN3270 user         */
   AP_UINT32        num_sessions;        /* number of sessions to delete   */
   unsigned char    delete_options;      /* delete all sessions / delete   */
                                         /*  user?                         */
} DELETE_TN3270_ACCESS;

typedef struct tn3270_session_name
{
   AP_UINT16         port_number;        /* TCP/IP port num of session     */
                                         /*  to delete                     */
} TN3270_SESSION_NAME;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_TN3270_ACCESS

default_record

Specifies whether this verb refers to the default

TN3270 user record that is used by any TN3270 user not explicitly identified by a TCP/IP address (deleting this record means that such users cannot access TN server). Possible values are:

AP_YES

This verb refers to the default TN3270 user record. The *client_address* parameter is reserved.

AP_NO

This verb refers to a normal TN3270 user record.

client_address

The TCP/IP address of the TN3270 user to be deleted. This is a string of 1-64 characters; the remaining bytes in the string should be set to nulls.

The address can be specified as a dotted-decimal IP address (such as 193.1.11.100), as a fully qualified name (such as newbox.this.co.uk), or as an alias (such as newbox), as specified on the DEFINE_TN3270_ACCESS verb.

num_sessions

The number of sessions to be deleted, as follows:

- To delete one or more of the user's sessions but leave other sessions configured, specify the number of sessions that are being deleted. Each of these must be defined by its TCP/IP port number, as described below.
- To delete all sessions, or to delete the user, specify zero in this parameter and do not include any TCP/IP port numbers. Specify the type of deletion required in the *delete_options* parameter below.

delete_options

If the *num_sessions* parameter (see above) is nonzero, this parameter is ignored. If *num_sessions* is zero, specify one of the following values:

AP_ALL_SESSIONS

Delete all sessions but leave the TN3270 user

configured.

AP_DELETE_USER

Delete the user and all the user's sessions.

For each session to be deleted, up to the number specified in *num_sessions*, the following parameter is required:

tn3270_session_name.port_number

The TCP/IP port number used for the session. This must match a port number defined for this TN3270 user.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CLIENT_ADDRESS

The specified client address did not match the TCP/IP address defined for any TN3270 user.

AP_INVALID_PORT_NUMBER

The specified TCP/IP port number did not match any TCP/IP port number defined for this user.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_TN3270_ASSOCIATION

DELETE_TN3270_ASSOCIATION deletes an association between a display LU and a printer LU, given the display LU name.

VCB Structure

```
typedef struct delete_tn3270_association
{
   AP_UINT16        opcode;               /* verb operation code          */
   unsigned char    reserv2;              /* reserved                     */
   unsigned char    format;               /* reserved                     */
   AP_UINT16        primary_rc;           /* primary return code          */
   AP_UINT32        secondary_rc;         /* secondary return code        */
   unsigned char    display_lu_name[8];   /* Display LU name              */
} DELETE_TN3270_ASSOCIATION;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_TN3270_ASSOCIATION

display_lu_name

Specifies the name of the display LU whose association is to be deleted. This is an EBCDIC string padded on the right with EBCDIC spaces.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LU_NAME

The display LU name was not a valid EBCDIC string.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_LU_NAME

No association is defined for the specified display LU.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_TP

DELETE_TP deletes a TP definition.

VCB Structure

```
typedef struct delete_tp
{
   AP_UINT16        opcode;                 /* verb operation code         */
   unsigned char    reserv2;                /* reserved                    */
   unsigned char    format;                 /* reserved                    */
   AP_UINT16        primary_rc;             /* primary return code         */
   AP_UINT32        secondary_rc;           /* secondary return code       */
   unsigned char    tp_name[64];            /* TP name                     */
} DELETE_TP;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DELETE_TP

tp_name Name of the TP to be deleted.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_TP_NAME

The *tp_name* parameter did not match the name of a

defined TP.

AP_SYSTEM_TP_CANT_BE_DELETED

The specified TP name is the name of a TP used internally by SNAplus2 you cannot delete it.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_TP_LOAD_INFO

The DELETE_TP_LOAD_INFO verb deletes a TP load information entry.

VCB Structure

```
typedef struct delete_tp_load_info
{
   AP_UINT16          opcode;                /* verb operation code        */
   unsigned char      reserv2;               /* reserved                   */
   unsigned char      format;                /* reserved                   */
   AP_UINT16          primary_rc;            /* primary return code        */
   AP_UINT32          secondary_rc;          /* secondary return code      */
   unsigned char      tp_name[64];           /* TP name                    */
   unsigned char      lu_alias[8];           /* LU alias                   */
} DELETE_TP_LOAD_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_DELETE_TP_LOAD_INFO
tp_name	The TP name of the TP load info entry to be deleted. This is a 64-byte EBCDIC string, padded on the right with spaces if the name is shorter than 64 characters.
lu_alias	The LU alias of the TP load info entry to be deleted. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_TP_NAME

The *tp_name* parameter did not match the name of a defined TP.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias specified for a TP load info entry for the TP name specified.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DELETE_USERID_PASSWORD

DELETE_USERID_PASSWORD deletes a password associated with a user ID, or removes profiles for a user ID and password.

VCB Structure

```
typedef struct delete_userid_password
{
   AP_UINT16               opcode;            /* verb operation code        */
   unsigned char           reserv2;           /* reserved                   */
   unsigned char           format;            /* reserved                   */
   AP_UINT16               primary_rc;        /* primary return code        */
   AP_UINT32               secondary_rc;      /* secondary return code      */
   AP_UINT16               delete_type;       /* type of delete             */
   unsigned char           user_id[10];       /* user id                    */
   USERID_PASSWORD_CHARS   password_chars;    /* password characteristics   */
} DELETE_USERID_PASSWORD;

typedef struct userid_password_chars
{
   unsigned char           description[32];    /* resource description      */
   unsigned char           reserv2[16];       /* reserved                   */
   AP_UINT16               profile_count;     /* number of profiles         */
   AP_UINT16               reserv1;           /* reserved                   */
   unsigned char           password[10];      /* password                   */
   unsigned char           profiles[10][10];  /* profiles                   */
} USERID_PASSWORD_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_DELETE_USERID_PASSWORD

delete_type

Specifies how this verb is being used. Possible values are:

AP_REMOVE_USER

Delete the user, password, and all associated profiles.

AP_REMOVE_PROFILES

Delete the specified profiles.

user_id

User identifier. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces if the name is shorter than 10 characters.

password_chars.description

This parameter is ignored.

password_chars.profile_count

Number of profiles to be deleted. If *delete_type* is set to AP_REMOVE_USER, this parameter is reserved.

password_chars.password

This parameter is ignored.

password_chars.profiles

Profiles associated with user. Each of these is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces if the profile name is shorter than 10 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_NO_PROFILES

The *delete_type* parameter was set to AP_REMOVE_PROFILES, but no profiles were specified.

AP_UNKNOWN_USER

The *user_id* parameter did not match a defined user ID.

AP_INVALID_UPDATE_TYPE

The *delete_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

DISCONNECT_NODE

An application uses this verb to release its handle to a SNAplus2 node when it has finished issuing NOF verbs to the node. The node from which the application wishes to disconnect is identified by the *target_handle* parameter on the call. After the verb completes successfully, the target handle identifying the node is no longer valid.

The application should always issue DISCONNECT_NODE for any open node handles before it exits, to allow SNAplus2 to free the resources associated with the application.

This verb may be issued either to a running node or to a server where the node is not running. It may also be issued to the local HP-UX client computer, if the application has used CONNECT_NODE to access the local client computer in order to control client-server tracing. For more information, see “CONNECT_NODE”.

VCB Structure

```
typedef struct disconnect_node
{
     AP_UINT16           opcode;            /* Verb operation code          */
     unsigned char       reserv2;           /* reserved                     */
     unsigned char       format;            /* reserved                     */
     AP_UINT16           primary_rc;        /* Primary return code          */
     AP_UINT32           secondary_rc;      /* Secondary return code        */
} DISCONNECT_NODE;
```

Supplied Parameters

opcode AP_DISCONNECT_NODE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_VERB_IN_PROGRESS

The specified target handle cannot be released because a previous verb issued for this handle is still outstanding. All verbs for the target handle must be completed before attempting to disconnect from the node.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

INIT_NODE

This verb starts a previously-defined node. The application must first issue CONNECT_NODE to obtain a target handle for the node; it then uses this target handle on the INIT_NODE call to identify the node to start.

This verb must be issued to a server where the node is not running.

VCB Structure

```
typedef struct init_node
{
     AP_UINT16           opcode;           /* verb operation code          */
     unsigned char       reserv2;          /* reserved                     */
     unsigned char       format;           /* reserved                     */
     AP_UINT16           primary_rc;       /* primary return code          */
     AP_UINT32           secondary_rc;     /* secondary return code        */
} INIT_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_INIT_NODE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter check, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_NODE_NAME

The node name specified in the configuration file does not match the name of the SNAplus2 computer to which the verb was issued.

AP_NOT_SERVER

The node name specified in the configuration file matches the name of the SNAplus2 computer, but the specified computer is a client (not a server) and cannot run the node.

AP_DLUR_NOT_SUPPORTED

The configuration of the node specifies that DLUR is supported, but the node is defined as a LEN node. DLUR cannot be supported on a LEN node.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_NODE_ALREADY_STARTED

The specified node has already been started.

AP_RESOURCE_NOT_LOADED

The node was not started because SNAplus2 detected one or more errors while attempting to load its configuration. Check the error log file for messages giving more details of the errors.

AP_INVALID_VERSION

The node was not started because there was a version mismatch between components of the SNAplus2 software. If you have upgraded your SNAplus2 license to include additional functions or users, check that you are using the correct version of the licensing software.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

INITIALIZE_SESSION_LIMIT

The INITIALIZE_SESSION_LIMIT verb initializes the session limits for a combination of local LU, partner LU, and mode.

You must issue this verb before you issue an ACTIVATE_SESSION verb.

VCB Structure

```
typedef struct initialize_session_limit
{
   AP_UINT16        opcode;                  /* verb operation code         */
   unsigned char    reserv2;                 /* reserved                    */
   unsigned char    format;                  /* reserved                    */
   AP_UINT16        primary_rc;              /* primary return code         */
   AP_UINT32        secondary_rc;            /* secondary return code       */
   unsigned char    lu_name[8];              /* local LU name               */
   unsigned char    lu_alias[8];             /* local LU alias              */
   unsigned char    plu_alias[8];            /* partner                     */
   unsigned char    fqplu_name[17];          /* fully qualified partner     */
                                             /*  LU name                    */
   unsigned char    reserv3;                 /* reserved                    */
   unsigned char    mode_name[8];            /* mode name                   */
   unsigned char    reserv3a;                /* reserved                    */
   unsigned char    set_negotiable;          /* set max negotiable limit?   */
   AP_UINT16        plu_mode_session_limit;  /* session limit               */
   AP_UINT16        min_conwinners_source;   /* minimum source contention   */
                                             /*  winner sessions            */
   AP_UINT16        min_conwinners_target;   /* minimum target contention   */
                                             /*  winner sessions            */
   AP_UINT16        auto_act;                /* auto activation limit       */
   unsigned char    reserv4[4];              /* reserved                    */
   AP_UINT32        sense_data;              /* sense data                  */
} INITIALIZE_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_INITIALIZE_SESSION_LIMIT

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

plu_alias

LU alias of the partner LU. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the partner LU is defined by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. This parameter is used only if the *plu_alias* field is set to zeros; it is ignored if *plu_alias* is specified.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Name of the mode to be used by the LUs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

set_negotiable

Specifies whether the maximum negotiable session limit for this mode, as defined by DEFINE_MODE, should be modified. Possible values are:

AP_YES

Use the value specified by *plu_mode_session_limit* as the maximum negotiable session limit for this LU-LU-mode combination.

AP_NO

Leave the maximum negotiable session limit as the value specified for the mode.

plu_mode_session_limit

Requested total session limit for this LU-LU-mode combination: the maximum number of parallel sessions permitted between these two LUs using this mode. Specify a value in the range 1-32,767. This value may be negotiated with the partner LU.

min_conwinners_source

Minimum number of sessions using this mode for which the local LU is the contention winner. Specify a value in the range 0-32,767. The sum of the *min_conwinners_source* and *min_conwinners_target* parameters must not exceed the *plu_mode_session_limit* parameter.

min_conwinners_target

Minimum number of sessions using this mode for which the partner LU is the contention winner. Specify a value in the range 0-32,767. The sum of the *min_conwinners_source* and *min_conwinners_target* parameters must not exceed the *plu_mode_session_limit* parameter.

auto_act

Number of sessions to activate automatically. Specify a value in the range 0-32,767. The actual number of automatically activated sessions is the minimum of this value and the negotiated minimum number of

contention winner sessions for the local LU.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Possible values are:

AP_AS_NEGOTIATED

The session limits were initialized, but one or more values were negotiated by the partner LU.

AP_AS_SPECIFIED

The session limits were initialized as requested, without being negotiated by the partner LU.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_EXCEEDS_MAX_ALLOWED

Indicates one of the following:

- The *plu_mode_session_limit*, *min_conwinners_source*, *min_conwinners_target*, or *auto_act* parameter was set to a value outside the valid range.
- The *min_conwinners_source*, *min_conwinners_target*, or *auto_act* parameter was set to a value greater than the value of *plu_mode_session_limit*.

AP_CANT_CHANGE_TO_ZERO

The *plu_mode_session_limit* parameter cannot be set to zero using this verb; use RESET_SESSION_LIMIT instead.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined local LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined local LU name.

AP_INVALID_MODE_NAME

The *mode_name* parameter did not match any defined mode name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name.

AP_INVALID_SET_NEGOTIABLE

The *set_negotiable* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_MODE_NOT_RESET

One or more sessions are currently active for this LU-LU-mode combination. Use CHANGE_SESSION_LIMIT instead of INITIALIZE_SESSION_LIMIT to specify the limits.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Session Allocation Error

If the verb does not execute because of a session allocation error, SNAplus2 returns the following parameters:

primary_rc AP_ALLOCATION_ERROR

secondary_rc AP_ALLOCATION_FAILURE_NO_RETRY

A session could not be allocated because of a condition that requires corrective action. Check the *sense_data* parameter and any logged messages to determine the reason for the failure, and take any action required. Do not attempt to retry the verb until the condition has been corrected.

sense_data The SNA sense data associated with the allocation failure.

Returned Parameters: CNOS Processing Errors

If the verb does not execute because of an error, SNAplus2 returns the following parameters.

primary_rc AP_CONV_FAILURE_NO_RETRY

The session limits could not be initialized because of a condition that requires action (such as a configuration mismatch or a session protocol error). Check the SNAplus2 log file for information about the error condition, and correct it before retrying this verb.

primary_rc AP_CNOS_PARTNER_LU_REJECT

secondary_rc AP_CNOS_COMMAND_RACE_REJECT

The verb failed because the specified mode was being accessed by another administration program (or internally by the SNAplus2 software) for session activation or deactivation, or for session limit processing. The application should retry the verb, preferably after a timeout to allow the race condition to be cleared.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of

primary and secondary return codes that are common to all NOF verbs.

OPEN_FILE

An application uses this verb to access the SNAplus2 domain configuration file in order to manage domain resources, or to access the `sna.net` file in order to manage backup master servers on the SNAplus2 LAN. The application should always issue CLOSE_FILE for any open file handles before it exits

This verb must be issued with a null target handle. If it completes successfully, SNAplus2 returns a handle identifying the file, which the application can then use on other NOF verbs to indicate the target for the verb.

VCB Structure

```
typedef struct open_file
{
    unsigned_s       opcode;              /* verb operation code           */
    unsigned char    reserv2;             /* reserved                      */
    unsigned char    format;              /* reserved                      */
    AP_UINT16        primary_rc;          /* primary return code           */
    AP_UINT32        secondary_rc;        /* secondary return code         */
    CONFIG_FILE      file_info;           /* definition of file requested  */
    AP_UINT32        target_handle;       /* handle for subsequent verbs   */
} OPEN_FILE;

typedef struct config_file
{
    unsigned char    requested_role;      /* config file requested         */
    unsigned char    role_supplied;       /* config file returned          */
    unsigned char    system_name[64];     /* computer name where file      */
                                          /*  located                      */
    unsigned char    file_name[81];       /* file name                     */
} CONFIG_FILE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_OPEN_FILE

file_info.requested_role

The type of file to be opened. Possible values are:

AP_MASTER

Open the master copy of the domain configuration file. This value must be used if the application intends to issue verbs that modify the configuration of domain resources.

AP_BACKUP

Open the master copy of the domain configuration file if available, otherwise a backup copy. This value may be used if the application intends to issue only QUERY_* verbs; if it needs to modify the configuration, it must use AP_MASTER, because it will not be able to obtain write access to a backup configuration file.

AP_SNA_NET

Open the sna.net file on the master server.

AP_TP_LOAD_INFO

Open a connection to the file on the local machine that contains information about how to load transaction programs (TPs).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

target_handle

Returned value for use on subsequent verbs directed to this file.

file_info.role_supplied

If *requested_role* was set to AP_BACKUP, this parameter indicates whether the file handle returned is for the master configuration file or a backup file. Possible values are:

AP_MASTER

Master configuration file.

AP_BACKUP

Backup configuration file.

For all other values of *requested_role*, this parameter is undefined.

file_info.system_name

Name of the SNAplus2 computer where the file is located.

file_info.file_name

Name of the file. This parameter is an ASCII string of 1-80 characters, followed by a null (0x00) character.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_FILE_NAME

The *file_name* parameter did not specify a valid configuration file name.

AP_INVALID_FILE_INFO

One of the parameters in the file_info structure was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_CONNECTION_NOT_MADE

SNAplus2 could not set up the local communications path to the file.

AP_FILE_BAD_RECORD

SNAplus2 detected an error in the configuration file. Check the error log file for a message giving more details of the error.

AP_FILE_ROLE_UNAVAILABLE

The application requested a master or backup configuration file, or the sna.net file, but no master or backup server was available. This is normally a temporary condition, occurring when a new server is taking over as master.

If the application is registered to receive server indications, it can check the *flags* parameter on these indications to determine when a new server has successfully taken over as master, and then retry the OPEN_FILE verb. For more information, see "SERVER_INDICATION". Alternatively, it can simply retry OPEN_FILE at intervals until it succeeds.

AP_INVALID_VERSION

One of the following has occurred:

- The SNAplus2 version number in the configuration file header does not match the version of the SNAplus2 software you are using. Check that you have the correct file; if the file was created using an earlier version of SNAplus2, refer to the *HP-UX SNAplus2 Upgrade Guide* for information about upgrading it.
- There was a version mismatch between components

of the SNAplus2 software. If you have upgraded your SNAplus2 license to include additional functions or users, check that you are using the correct version of the licensing software.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

4 NOF API Verbs (QUERY Verbs)

QUERY_3270_DIAG

QUERY_3270_DIAG returns information about the 3270 diagnostics parameters for the SNAplus2 system. It always returns information about the parameters used to record and display response-time data; depending on the options used, it can also return data on a 3270 user alert that users can send to the host NetView program, or on multiple alerts.

This verb must be issued to the domain configuration file. If 3270 diagnostics parameters have not been defined, the verb returns the default parameters that SNAplus2 uses for response-time data; no alert definitions are returned.

VCB Structure

```
typedef struct query_3270_diag
{
    AP_UINT16           opcode;               /* verb operation code        */
    unsigned char       reserv2;              /* reserved                   */
    unsigned char       format;               /* reserved                   */
    AP_UINT16           primary_rc;           /* primary return code        */
    AP_UINT32           secondary_rc;         /* secondary return code      */
    unsigned char       *buf_ptr;             /* pointer to buffer          */
    AP_UINT32           buf_size;             /* buffer size                */
    AP_UINT32           total_buf_size;       /* total buffer size required */
    AP_UINT16           num_entries;          /* number of entries          */
    AP_UINT16           total_num_entries;    /* total number of entries    */
    unsigned char       list_options;         /* listing options            */
    unsigned char       reserv3;              /* reserved                   */
    AP_UINT16           alert_number;         /* index into alerts          */
    DIAG_3270_DATA      def_data;
} QUERY_3270_DIAG;

typedef struct diag_3270_data
{
    unsigned char       rtm_overflow;         /* Send RTM data at counter   */
                                              /*  overflow                  */
    unsigned char       rtm_unbind;           /* Send RTM data at UNBIND    */
    unsigned char       rtm_timer_option;     /* RTM timers option          */
    unsigned char       reserv1;              /* reserved                   */
    AP_UINT16           rtm_thresh1;          /* RTM threshold #1           */
```

```
    AP_UINT16           rtm_thresh2;          /* RTM threshold #2          */
    AP_UINT16           rtm_thresh3;          /* RTM threshold #3          */
    AP_UINT16           rtm_thresh4;          /* RTM threshold #4          */
    AP_UINT16           num_alerts;           /* Number of user alerts     */
} DIAG_3270_DATA;

typedef struct alert_3270_data
{
    unsigned char       description[53];      /* description               */
    unsigned char       parameter1[33];       /* parameter 1               */
    unsigned char       parameter2[33];       /* parameter 2               */
    unsigned char       parameter3[33];       /* parameter 3               */
} ALERT_3270_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_3270_DIAG

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of 3270 user alerts for which data should be returned. To request data for a specific alert rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of alerts from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *alert_number* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *alert_number* parameter.

For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs".

If no alerts have been defined, this parameter must be set to AP_FIRST_IN_LIST.

alert_number

The number of the alert for which information is required, or the number to be used as an index into the list of alerts. This is a number in the range 1-20; it is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of alert entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of alert entries actually returned.

`diag_3270_data`

Data structure containing 3270 diagnostics information. The format of this information is the same as for the DEFINE_3270_DIAG verb.

For each alert, up to the number specified in *num_entries*, an `alert_3270_data` structure is included. The format of the information in this structure is the same as for the DEFINE_3270_DIAG verb.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_ALERT_NUMBER

The *list_options* parameter was set to AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT, but one of the following has occurred:

- The *alert_number* parameter was not in the valid range.
- The alert with the specified number has not been defined.
- No alerts have been defined.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_3270_USER

QUERY_3270_USER returns information about users of the SNAplus2 3270 emulation program. It can return either summary or detailed information, about a single user or multiple users, depending on the options used. This verb returns information about the user's current usage of 3270 emulation, not about the definition of this user in the configuration file; use QUERY_3270_USER_DEF to obtain information about the configuration file definition (such as session limits and access to 3270 functions), and QUERY_3270_USER_SESSIONS to obtain detailed information about individual sessions.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_3270_user
{
   AP_UINT16         opcode;              /* verb operation code           */
   unsigned char     reserv2;             /* reserved                      */
   unsigned char     format;              /* reserved                      */
   AP_UINT16         primary_rc;          /* primary return code           */
   AP_UINT32         secondary_rc;        /* secondary return code         */
   unsigned char     *buf_ptr;            /* pointer to buffer             */
   AP_UINT32         buf_size;            /* buffer size                   */
   AP_UINT32         total_buf_size;      /* total buffer size required    */
   AP_UINT16         num_entries;         /* number of entries             */
   AP_UINT16         total_num_entries;   /* total number of entries       */
   unsigned char     list_options;        /* listing options               */
   unsigned char     reserv3;             /* reserved                      */
   AP_UINT16         scope;               /* parameters to match on        */
   unsigned char     user_name[32];       /* 3270 user name                */
   unsigned char     system_name[64];     /* computer name                 */
   AP_UINT32         user_pid;            /* process ID                    */
} QUERY_3270_USER;


typedef struct user_3270_summary
{
   AP_UINT16    overlay_size;             /* size of returned entry        */
   unsigned char     user_name[32];       /* 3270 user name                */
   unsigned char     system_name[64];     /* computer name                 */
   AP_UINT32         user_pid;            /* process ID                    */
```

```
} USER_3270_SUMMARY;

typedef struct user_3270_detail
{
   AP_UINT16         overlay_size;         /* size of returned entry         */
   unsigned char     user_name[32];        /* 3270 user name                 */
   unsigned char     system_name[64];      /* computer name                  */
   AP_UINT32         user_pid;             /* process ID                     */
   AP_UINT32         reserv1;              /* reserved                       */
   AP_UINT32         user_uid;             /* user ID                        */
   AP_UINT32         user_gid;             /* group ID                       */
   unsigned char     user_gname[32];       /* group name                     */
   AP_UINT32         user_session_count;   /* count of user's 3270 sessions  */
   AP_UINT32         user_start_time;      /* time when user started the     */
                                           /*  3270 emulation program        */
} USER_3270_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_3270_USER

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of users for which data should be returned. To request data for a specific user rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2

will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *scope*, *user_name*, *system_name*, and *user_pid* parameters (see below).

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *scope*, *user_name*, *system_name*, and *user_pid* parameters (see below).

The *scope* parameter specifies whether to filter the returned information by user name, computer name, process ID, or any combination of these. For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

scope

The range of 3270 emulation users for which information should be returned. This is used in conjunction with the *list_options* parameter to specify which entries are required. Possible values are:

AP_USER

Return information about all 3270 emulation programs running with the specified user name, or on all 3270 users if no name is specified.

AP_SYSTEM

Return information about 3270 emulation programs running on the specified computer.

AP_USER_AT_SYSTEM

Return information about all 3270 emulation programs running with the specified 3270 user name on the specified computer.

AP_USER_PROCESS

Return information about a specific copy of the 3270 emulation program, identified by the 3270 user name, computer name, and process ID.

user_name

The name of the 3270 user for whom information is required, or the name to be used as an index into the list of users. The user name is an ASCII string of 1-32 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

system_name

The computer name for which 3270 user information is required; this is an ASCII string of 1-64 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

If you are running SNAplus2 with all programs on a single computer, you can set this parameter to all binary zeros; there is no need to specify the computer name. .

user_pid

The process ID of the 3270 emulation program for which information is required, or the process ID to be used as an index into the list of users. This parameter is ignored if *list_options* is set to

AP_FIRST_IN_LIST. If this parameter is specified and you are running SNAplus2 in a client-server configuration, the *system_name* parameter must also be specified.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned.

Each entry in the data buffer consists of the following:

user_3270_summary.overlay_size

The size of the returned user_3270_summary structure, and therefore the offset to the start of the next entry in the data buffer.

user_3270_summary.user_name

The name of the 3270 user. This is an ASCII string of 1-32 characters.

user_3270_summary.system_name

The computer name on which the 3270 emulation program is running.

user_3270_summary.user_pid

The process ID of the 3270 emulation program.

user_3270_detail.overlay_size

The size of the returned user_3270_detail structure, and therefore the offset to the start of the next entry in the data buffer.

user_3270_detail.user_name

The name of the 3270 user. This is an ASCII string of 1-32 characters.

user_3270_detail.system_name

The computer name on which the 3270 emulation program is running.

user_3270_detail.user_pid

The process ID of the 3270 emulation program.

user_3270_detail.user_uid

The HP-UX user ID with which the 3270 emulation program is running.

user_3270_detail.user_gid

The HP-UX group ID with which the 3270 emulation program is running.

user_3270_detail.user_gname

The HP-UX group name with which the 3270 emulation program is running.

user_3270_detail.user_session_count

The number of sessions currently active for this copy of the 3270 emulation program. The application can use the QUERY_3270_USER_SESSIONS verb to obtain detailed information about these sessions.

user_3270_detail.user_start_time

The time at which the user started the 3270 emulation program (this may be earlier than the start time for the first session). This value is specified as “seconds since epoch” (the number of seconds since the start of the

year 1970).

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_3270_USER

The *user_name*, *system_name*, or *user_pid* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_3270_USER_SESSIONS

QUERY_3270_USER returns information about a 3270 emulation user's current sessions. It can return either summary or detailed information, about a single session or multiple sessions, depending on the options used. The application can use QUERY_EMULATOR_USER_DEF to obtain information about the configuration file definition of a 3270 user (such as session limits and access to 3270 functions), and QUERY_3270_USER to obtain general information about the user's 3270 emulation program usage.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_3270_user_sessions
{
    AP_UINT16         opcode;              /* verb operation code          */
    unsigned char     reserv2;             /* reserved                     */
    unsigned char     format;              /* reserved                     */
    AP_UINT16         primary_rc;          /* primary return code          */
    AP_UINT32         secondary_rc;        /* secondary return code        */
    unsigned char     *buf_ptr;            /* pointer to buffer            */
    AP_UINT32         buf_size;            /* buffer size                  */
    AP_UINT32         total_buf_size;      /* total buffer size required   */
    AP_UINT16         num_entries;         /* number of entries            */
    AP_UINT16         total_num_entries;   /* total number of entries      */
    unsigned char     list_options;        /* listing options              */
    unsigned char     reserv3;             /* reserved                     */
    unsigned char     user_name[32];       /* 3270 user name               */
    unsigned char     system_name[64];     /* computer name                */
    AP_UINT32         user_pid;            /* process ID                   */
    unsigned char     lu_name[8];          /* LU of session to start from  */
} QUERY_3270_USER_SESSIONS;


typedef struct user_3270_session_summary
{
    AP_UINT16         overlay_size;        /* size of returned entry       */
    unsigned char     lu_name[8];          /* LU used by session           */
} USER_3270_SESSION_SUMMARY;


typedef struct user_3270_session_detail
```

```
{
    AP_UINT16         overlay_size;         /* size of returned entry        */
    unsigned char     lu_name[8];           /* LU used by session            */
    unsigned char     nau_address;          /* NAU adddress of LU            */
    AP_UINT32         reserv1[3];           /* reserved                      */
} USER_3270_SESSION_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_3270_USER_SESSIONS

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of sessions for which data should be returned. To request data for a specific session rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of sessions from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first session in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *lu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *lu_name* parameter.

For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs".

user_name

The name of the 3270 user for whom information is required; this is an ASCII string of 1-32 characters. This parameter must be specified.

system_name

The computer name for which 3270 user information is required; this is an ASCII string of 1-64 characters. .

If the SNAplus2 system includes only one server, there is no need to specify the computer name (it can be left as all binary zeros). If there are multiple servers and no computer name is specified, SNAplus2 returns the first entry it finds for the specified user name; there is no need to specify the computer name unless the specified user is using the 3270 emulation program on more than one computer.

user_pid

The process ID of the 3270 emulation program for which information is required.

If the SNAplus2 system includes only one server, this parameter is required only if the user is using more than one copy of the 3270 emulation program. If there are multiple servers and no process ID is specified, SNAplus2 returns the first entry it finds for the specified user name; there is no need to specify the process ID unless the specified user is using more than one copy of the 3270 emulation program on the computer specified by the *system_name* parameter. If the process ID is not required, set it to zero.

lu_name

The LU used by the session for which information is required, or the LU name to be used as an index into the list of sessions. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-character EBCDIC string, padded on the right with EBCDIC spaces if necessary.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned.

system_name

The computer name on which the 3270 emulation program is running.

user_pid

The process ID of the 3270 emulation program.

Each entry in the data buffer consists of the following:

user_3270_session_summary.overlay_size

The size of the returned user_3270_session_summary structure, and therefore the offset to the start of the next entry in the data buffer.

user_3270_session_summary.lu_name

The LU being used by the session.

user_3270_session_detail.overlay_size

The size of the returned user_3270_session_detail structure, and therefore the offset to the start of the next entry in the data buffer.

user_3270_session_detail.lu_name

The LU being used by the session.

user_3270_session_detail.nau_address

The NAU address of the LU used by the 3270 session.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_3270_USER

The *user_name*, *system_name*, or *user_pid* parameter was not set to a valid value.

AP_INVALID_3270_SESSION

The *lu_name* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_ACTIVE_TRANSACTION

QUERY_ACTIVE_TRANSACTION returns information about active Multiple Domain Support (MDS) transactions known to the SNAplus2 Management Services component. An active transaction is an MDS request for which a reply has not yet been received.

This verb may be used to obtain information about a single transaction, or on multiple transactions, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_active_transaction
{
   AP_UINT16        opcode;                 /* verb operation code          */
   unsigned char    reserv2;                /* reserved                     */
   unsigned char    format;                 /* reserved                     */
   AP_UINT16        primary_rc;             /* primary return code          */
   AP_UINT32        secondary_rc;           /* secondary return code        */
   unsigned char   *buf_ptr;                /* pointer to buffer            */
   AP_UINT32        buf_size;               /* buffer size                  */
   AP_UINT32        total_buf_size;         /* total buffer size required   */
   AP_UINT16        num_entries;            /* number of entries            */
   AP_UINT16        total_num_entries;      /* total number of entries      */
   unsigned char    list_options;           /* listing options              */
   unsigned char    reserv3;                /* reserved                     */
   unsigned char    fq_req_loc_cp_name[17]; /* fq cp name of transaction    */
                                            /*  requestor                   */
   unsigned char    req_agent_appl_name[8]; /* appl name of transaction     */
                                            /*  requestor                   */
   unsigned char    seq_num_dt[17];         /* sequence number date/time    */
} QUERY_ACTIVE_TRANSACTION;

typedef struct active_transaction_data
{
   AP_UINT16       overlay_size;            /* size of returned entry       */
   unsigned char   fq_origin_cp_name[17];   /* cp name of transaction origin */
   unsigned char   origin_ms_appl_name[8];  /* appl name of transaction     */
                                            /*  origin                      */
   unsigned char   fq_dest_cp_name[17];     /* cp name of transaction       */
                                            /*  destination                 */
   unsigned char   dest_ms_appl_name[8];    /* appl name of transaction dest */
```

```
   unsigned char   fq_req_loc_cp_name[17];  /* fq cp name of transaction     */
                                            /*  requestor                    */
   unsigned char   req_agent_appl_name[8];  /* appl name of transaction      */
                                            /*  requestor                    */
   unsigned char   seq_num_dt[17];          /* sequence number date/time     */
   unsigned char   reserva[20];             /* reserved                      */
} ACTIVE_TRANSACTION_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_ACTIVE_TRANSACTION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()`function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of transactions for which data should be returned. To request data for a specific transaction rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *fq_req_loc_cp_name*, *req_agent_appl_name*, and *seq_num_dt* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *fq_req_loc_cp_name*, *req_agent_appl_name*, and *seq_num_dt* parameters.

The list is ordered by *fq_req_loc_cp_name*, then by *req_agent_appl_name*, and finally in numerical order of *seq_num_dt*. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

fq_req_loc_cp_name

Fully qualified control point name of the transaction requestor. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

req_agent_appl_name

Application name of the transaction requestor. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

This name is normally an EBCDIC string, using type-1134 characters (uppercase A-Z and numerals 0-9); alternatively, it can be one of the MS Discipline-Specific Application Programs specified in *SNA Management Services Reference*. The string must be 8 characters long; pad on the right with EBCDIC space characters (0x40) if necessary.

seq_num_dt

Sequence number date/time correlator (17 bytes long)

of the original transaction, as defined in the IBM SNA Formats manual. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

active_transaction_data.overlay_size

The size of the returned active_transaction_data structure, and therefore the offset to the start of the next entry in the data buffer.

active_transaction_data.fq_origin_cp_name

Fully qualified control point name of the origin for the transaction. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC

dot (period) character, and a network name of up to 8 A-string characters.

active_transaction_data.origin_ms_appl_name

Application name of the origin for the transaction. This name is normally an 8-character EBCDIC string, using type-1134 characters (uppercase A-Z and numerals 0-9); alternatively, it can be one of the MS Discipline-Specific Application Programs specified in *Systems Network Architecture: Management Services Reference* (see "Related Publications").

active_transaction_data.fq_dest_cp_name

Fully qualified control point name of the destination for the transaction. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

active_transaction_data.dest_ms_appl_name

Application name of the destination application for the transaction. This name is normally an 8-character EBCDIC string, using type-1134 characters (uppercase A-Z and numerals 0-9); alternatively, it can be one of the MS Discipline-Specific Application Programs specified in *Systems Network Architecture: Management Services Reference* (see "Related Publications").

active_transaction_data.fq_req_loc_cp_name

Fully qualified control point name of the transaction requestor. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

active_transaction_data.req_agent_appl_name

Application name of the transaction requestor. This name is normally an 8-character EBCDIC string, using type-1134 characters (uppercase A-Z and numerals

0-9); alternatively, it can be one of the MS Discipline-Specific Application Programs specified in *Systems Network Architecture: Management Services Reference* (see "Related Publications").

active_transaction_data.seq_num_dt

Sequence number date/time correlator (17 bytes long) of the original transaction, as defined in the IBM SNA Formats manual.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_ACTIVE_TRANSACTION

The control point name, application name, or sequence number correlator did not match that of an active transaction.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_AVAILABLE_TP

QUERY_AVAILABLE_TP returns information about active invokable TPs (APPC applications that have issued the RECEIVE_ALLOCATE verb, or CPI-C applications that have issued the Accept_Conversation or Accept_Incoming call). This verb can be used to obtain information about a specific TP or about multiple TPs, depending on the options used. This verb returns information about all such TPs that are running, whether or not they currently have an APPC verb or CPI-C call outstanding to accept a new incoming conversation.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_available_tp
{
    AP_UINT16         opcode;             /* Verb operation code           */
    unsigned char     reserv2;            /* reserved                      */
    unsigned char     format;             /* reserved                      */
    AP_UINT16         primary_rc;         /* Primary return code           */
    AP_UINT32         secondary_rc;       /* Secondary return code         */
    unsigned char     *buf_ptr;           /* pointer to buffer             */
    AP_UINT32         buf_size;           /* buffer size                   */
    AP_UINT32         total_buf_size;     /* total buffer size required    */
    AP_UINT16         num_entries;        /* number of entries             */
    AP_UINT16         total_num_entries;  /* total number of entries       */
    unsigned char     list_options;       /* listing options               */
    unsigned char     reserv3;            /* reserved                      */
    unsigned char     tp_name[64];        /* TP name                       */
    unsigned char     system_name[64];    /* computer name where TP is     */
                                          /*  running                      */
} QUERY_AVAILABLE_TP;


typedef struct available_tp_data
{
    AP_UINT16         overlay_size;       /* size of returned entry        */
    unsigned char     tp_name[64];        /* TP name                       */
    unsigned char     system_name[64];    /* computer name where TP is     */
                                          /*  running                      */
} AVAILABLE_TP_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_AVAILABLE_TP

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()`function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of TPs for which data should be returned. To request data for a specific TP rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of TPs from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of TP name and system name.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of TP name and system

name.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

system_name

The computer name for which TP information is required. The system name is an ASCII string of 1-64 characters, which must match a SNAplus2 computer name. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

For the standalone version of SNAplus2, there is no need to specify the computer name (it can be left as all binary zeros). For the client-server version, specify the computer name to list only TPs on the specified computer, or leave it as all binary zeros to list TPs on all computers.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

available_tp_data.overlay_size

The size of the returned available_tp_data structure, and therefore the offset to the start of the next entry in the data buffer.

available_tp_data.tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters.

available_tp_data.system_name

Name of the computer where the TP is running.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

AP_UNKNOWN_TP

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *tp_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all

NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_BCK_CS_TRACE

This verb returns information about the current options for back-level client-server tracing (tracing of data sent between a current-level SNAplus2 server and a back-level server or client). It is used only when you are in the processs of migrating a client-server SNAplus2 system to a new release of the software, so that one or more servers are running the current-level software and providing support for computers running the back-level software. For more information about the migration process and on support for back-level computers, see the *HP-UX SNAplus2 Upgrade Guide*.

This verb must be issued to a running node on a computer where the back-level support software is running (for more information, see the *HP-UX SNAplus2 Upgrade Guide*).

VCB Structure

```
typedef struct query_bck_cs_trace
{
     AP_UINT16          opcode;             /* verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* primary return code          */
     AP_UINT32          secondary_rc;       /* secondary return code        */
     AP_UINT32          trace_flags;        /* trace flags                  */
} QUERY_BCK_CS_TRACE;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_BCK_CS_TRACE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

trace_flags The types of tracing currently active. If no tracing is active, this is set to AP_BCK_NO_TRACE.

If tracing is being used on specific interfaces, this parameter is set to one or more values from the list below, combined using a logical OR operation.

AP_BCK_UDP_TRACE

UDP messages

AP_BCK_TCP_TRACE

TCP messages

AP_BCK_UDP_TEXT

Text strings describing UDP events

AP_BCK_TCP_TEXT

Text strings describing TCP events

AP_BCK_TEXT

Text strings describing other events

AP_BCK_MS_TRACE

Tracing on TRANSFER_MS_DATA verbs (passed to the MS library on the server)

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_BUFFER_AVAILABILITY

This verb returns information about the amount of STREAMS buffers that SNAplus2 is currently using, the maximum amount it has used, and the maximum amount available (specified using the SET_BUFFER_AVAILABILITY verb). This allows you to check STREAMS buffer usage and set the limit appropriately, to ensure that sufficient buffers are available for SNAplus2 components and for other programs on the HP-UX computer. The verb also returns additional internal values relating to buffer usage, for use by SNAplus2 support personnel.

VCB Structure

```
typedef struct query_buffer_availability
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* primary return code          */
   AP_UINT32        secondary_rc;          /* secondary return code        */
   AP_UINT32        reset_max_values;      /* set stored max values to     */
                                           /*  current                     */
   AP_UINT32        buf_avail;             /* maximum buffer space         */
                                           /*  available                   */
   AP_UINT32        buf_total_count;       /* current buffer usage - count  */
   AP_UINT32        buf_total_bytes;       /* current buffer usage - bytes  */
   AP_UINT32        buf_rsrv_count;        /* buffers reserved - count     */
   AP_UINT32        buf_rsrv_bytes[2];     /* buffers reserved - bytes     */
   AP_UINT32        buf_res_use_count;     /* usage of reserved buffers    */
                                           /*  - count                     */
   AP_UINT32        buf_res_use_bytes;     /* usage of reserved buffers    */
                                           /*  - bytes                     */
   AP_UINT32        peak_usage;            /* peak usage                   */
   AP_UINT32        peak_decay;            /* peak decay                   */
   unsigned char    throttle_status;       /* throttle status              */
   unsigned char    buf_use_status;        /* congestion status            */
   AP_UINT32        max_buf_total_count;   /* maximum buffer usage - count  */
   AP_UINT32        max_buf_total_bytes;   /* maximum buffer usage - bytes  */
   AP_UINT32        max_buf_rsrv_count;    /* max buffers reserved - count  */
   AP_UINT32        max_buf_rsrv_bytes[2]; /* max buffers reserved - bytes  */
   AP_UINT32        max_buf_res_use_count; /* max rsrv buffer usage - count */
   AP_UINT32        max_buf_res_use_bytes; /* max rsrv buffer usage - bytes */
```

```
   AP_UINT32        max_peak_usage;         /* maximum peak usage           */
   unsigned char    max_throttle_status;    /* maximum throttle status      */
   unsigned char    max_buf_use_status;     /* maximum congestion status    */
   unsigned char    debug_param[32];        /* reserved                     */
} QUERY_BUFFER_AVAILABILITY;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_BUFFER_AVAILABILITY

reset_max_values

Specify whether SNAplus2 should reset the values for the *max_** parameters (after returning them on this verb) to match the current values of these parameters. This ensures that a subsequent QUERY_BUFFER_AVAILABILITY verb will return the maximum values reached since this verb, rather than the maximum values reached since the system was started (or since the values for the *max_** parameters were last reset). Possible values are:

AP_YES

Reset the values for the *max_** parameters to match the current values.

AP_NO

Do not reset the values for the *max_** parameters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters. Values returned in other fields are for use by SNAplus2 support personnel.

primary_rc

AP_OK

secondary_rc

Not used.

buf_avail

The maximum amount of STREAMS buffer space available to SNAplus2, in bytes, as defined by a SET_BUFFER_AVAILABILITY verb.

buf_total_count

The number of buffers currently allocated to SNAplus2 components.

buf_total_bytes

The total amount of storage in buffers currently allocated to SNAplus2 components.

buf_rsrv_count

The total number of buffers reserved.

buf_rsrv_bytes

The total amount of storage in buffers reserved, in bytes.

buf_res_use_count

The number of reserved buffers in use.

buf_res_use_bytes

The number of bytes in the reserved buffers currently in use.

peak_usage

Peak buffer usage-smoothed percentage of buffers that are actually used.

peak_decay

Smoothing parameter.

throttle_status

Adaptive pacing status.

buf_use_status

Congestion status. Possible values are:

```
AP_CONGESTED
```

```
AP_UNCONGESTED
```

max_buf_total_count

The maximum number of buffers that have been allocated to SNAplus2 components at any time.

max_buf_total_bytes

The maximum amount of buffer storage that has been allocated to SNAplus2 components at any time.

max_buf_rsrv_count

The maximum number of buffers that can be reserved.

max_buf_rsrv_bytes

The maximum amount of buffer storage that can be reserved, in bytes.

max_buf_res_use_count

The maximum number of reserved buffers that can be in use.

max_buf_res_use_bytes

The maximum number of bytes of reserved buffers that can be in use at any time.

max_peak_usage

Maximum peak buffer usage-smoothed percentage of buffers actually used.

max_throttle_status

Maximum adaptive pacing status.

max_buf_use_status

Maximum congestion status. Possible values are:

```
AP_CONGESTED
AP_UNCONGESTED
```

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CENTRAL_LOGGER

This verb returns the name of the node currently defined as the central logger (the node holding the central log file to which SNAplus2 log messages from all servers are sent). This verb does not return information about whether central logging is active; use QUERY_CENTRAL_LOGGING to determine this.

This verb must be issued with a null target handle.

VCB Structure

```
typedef struct query_central_logger
{
     AP_UINT16          opcode;             /* verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* primary return code          */
     AP_UINT32          secondary_rc;       /* secondary return code        */
     unsigned char      node_name[64];      /* name of central logger       */
} QUERY_CENTRAL_LOGGER;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_CENTRAL_LOGGER

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

node_name The name of the node defined as the central logger.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns

the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_NO_CENTRAL_LOG

No master server is currently active.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CENTRAL_LOGGING

This verb returns information about whether SNAplus2 log messages are sent to a central file from all servers, or to a separate file on each server. For more information, see "SET_LOG_FILE".

This verb must be issued to the node that is currently acting as the central logger; for information about accessing this node, see "CONNECT_NODE".

VCB Structure

```
typedef struct query_central_logging
{
     AP_UINT16          opcode;           /* verb operation code             */
     unsigned char      reserv2;          /* reserved                        */
     unsigned char      format;           /* reserved                        */
     AP_UINT16          primary_rc;       /* primary return code             */
     AP_UINT32          secondary_rc;     /* secondary return code           */
     unsigned char      enabled;          /* is central logging enabled?     */
} QUERY_CENTRAL_LOGGING;
```

Supplied Parameters

The application supplies the following parameter:

opcode	AP_QUERY_CENTRAL_LOGGING

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK
secondary_rc	Not used.
enabled	Specifies whether central logging is enabled or disabled. Possible values are: AP_YES Central logging is enabled. All log messages are sent to

a single file on the node currently acting as the central logger.

AP_NO

Central logging is disabled. Log messages from each server are sent to a file on that server (specified using the SET_LOG_FILE verb).

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_NOT_CENTRAL_LOGGER

The verb was issued to a node that is not the central logger.

State Check

If the command does not execute because of a state error, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_NO_CENTRAL_LOG

No master server is currently active.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CN

QUERY_CN returns information about adjacent Connection Networks. This information is structured as “determined data” (data gathered dynamically during execution) and “defined data” (the data supplied by the application on DEFINE_CN).

This verb can be used to obtain information about a specific connection network, or about multiple connection networks, depending on the options used. It can be issued only at an end node; it is not valid at a LEN node.

VCB Structure

```
typedef struct query_cn
{
   AP_UINT16         opcode;             /* Verb operation code             */
   unsigned char     reserv2;            /* reserved                        */
   unsigned char     format;             /* reserved                        */
   AP_UINT16         primary_rc;         /* Primary return code             */
   AP_UINT32         secondary_rc;       /* Secondary return code           */
   unsigned char     *buf_ptr;           /* pointer to buffer               */
   AP_UINT32         buf_size;           /* buffer size                     */
   AP_UINT32         total_buf_size;     /* total buffer size required      */
   AP_UINT16         num_entries;        /* number of entries               */
   AP_UINT16         total_num_entries;  /* total number of entries         */
   unsigned char     list_options;       /* listing options                 */
   unsigned char     reserv3;            /* reserved                        */
   unsigned char     fqcn_name[17];      /* Name of Connection Network      */
} QUERY_CN;

typedef struct cn_data
{
   AP_UINT16         overlay_size;       /* size of returned entry          */
   unsigned char     fqcn_name[17];      /* Name of Connection Network      */
   unsigned char     reserv1;            /* reserved                        */
   CN_DET_DATA       det_data;           /* Determined data                 */
   CN_DEF_DATA       def_data;           /* Defined data                    */
} CN_DATA;

typedef struct cn_det_data
{
```

```
   AP_UINT16         num_act_ports;       /* number of active ports          */
   unsigned char     reserva[20];         /* reserved                        */
} CN_DET_DATA;


typedef struct cn_def_data
{
   unsigned char     description[32];     /* resource description            */
   unsigned char     reserve0[16];        /* reserved                        */
   unsigned char     num_ports;           /* number of ports on CN           */
   unsigned char     reserve1[16];        /* reserved                        */
   TG_DEFINED_CHARS  tg_chars;            /* TG characteristics              */
} CN_DEF_DATA;

typedef struct tg_defined_chars
{
   unsigned char     effect_cap;          /* effective capacity              */
   unsigned char     reserve1[5];         /* reserved                        */
   unsigned char     connect_cost;        /* connection cost                 */
   unsigned char     byte_cost;           /* byte cost                       */
   unsigned char     reserve2;            /* reserved                        */
   unsigned char     security;            /* security                        */
   unsigned char     prop_delay;          /* propagation delay               */
   unsigned char     modem_class;         /* reserved                        */
   unsigned char     user_def_parm_1;     /* user-defined parameter 1        */
   unsigned char     user_def_parm_2;     /* user-defined parameter 2        */
   unsigned char     user_def_parm_3;     /* user-defined parameter 3        */
} TG_DEFINED_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_QUERY_CN
overlay_size	For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof()function.
buf_ptr	A pointer to a data buffer that SNAplus2 will use to return the requested information.
buf_size	Size of the supplied data buffer.
num_entries	Maximum number of CNs for which data should be returned. To request data for a specific CN rather than

a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list of CNs from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *fqcn_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *fqcn_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

fqcn_name Fully qualified name of the CN for which information is required, or the name to be used as an index into the list of CNs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied

buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cn_data.overlay_size

The size of the returned cn_data structure, and therefore the offset to the start of the next entry in the data buffer.

cn_data.fqcn_name

Fully qualified name of the CN. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

cn_data.det_data.num_act_ports

The number of active ports on the connection network.

cn_data.def_data.description

A null-terminated text string describing the CN, as specified in the definition of the CN.

cn_data.def_data.num_ports

The total number of ports on the connection network.

cn_data.def_data.tg_chars.effect_cap

Actual bits per second rate (line speed). The value is

encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is eeeeemmm. Each unit of effective capacity is equal to 300 bits per second.

cn_data.def_data.tg_chars.connect_cost

Cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

cn_data.def_data.tg_chars.byte_cost

Cost per byte. Values are integers in the range 0-255, where zero is the lowest cost per byte and 255 is the highest.

cn_data.def_data.tg_chars.security

Security level of the network. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

AP_SEC_MAXIMUM

Maximum security.

cn_data.def_data.tg_chars.prop_delay

Propagation delay: the time that a signal takes to travel the length of the link. Possible values are:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

cn_data.def_data.tg_chars.user_def_parm_1 through *def_data.tg_chars.user_def_parm_3*

User-defined parameters, which include other TG characteristics not covered by the above parameters. Each of these parameters is set to a value in the range 0-255.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_CN_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *fqcn_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node is a LEN node, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node is a LEN node. This verb is valid only at an end node.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CN_PORT

QUERY_CN_PORT returns information about ports defined on an adjacent Connection Network.

This verb can be used to obtain information about a specific port, or about multiple ports, depending on the options used. It can be issued only at an end node; it is not valid at a LEN node.

VCB Structure

```
typedef struct query_cn_port
{
     AP_UINT16          opcode;             /* Verb operation code          */
     unsigned char      reserv2;            /* reserved                     */
     unsigned char      format;             /* reserved                     */
     AP_UINT16          primary_rc;         /* Primary return code          */
     AP_UINT32          secondary_rc;       /* Secondary return code        */
     unsigned char     *buf_ptr;            /* pointer to buffer            */
     AP_UINT32          buf_size;           /* buffer size                  */
     AP_UINT32          total_buf_size;     /* total buffer size required   */
     AP_UINT16          num_entries;        /* number of entries            */
     AP_UINT16          total_num_entries;  /* total number of entries      */
     unsigned char      list_options;       /* listing options              */
     unsigned char      reserv3;            /* reserved                     */
     unsigned char      fqcn_name[17];      /* Name of Connection Network   */
     unsigned char      port_name[8];       /* port name                    */
} QUERY_CN_PORT;

typedef struct cn_port_data
{
     AP_UINT16          overlay_size;       /* size of returned entry       */
     unsigned char      fqcn_name[17];      /* Name of Connection Network   */
     unsigned char      port_name[8];       /* name of port                 */
     unsigned char      tg_num;             /* transmission group number    */
     unsigned char      reserva[20];        /* reserved                     */
} CN_PORT_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_QUERY_CN_PORT
overlay_size	For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof()function.
buf_ptr	A pointer to a data buffer that SNAplus2 will use to return the requested information.
buf_size	Size of the supplied data buffer.
num_entries	Maximum number of ports for which data should be returned. To request data for a specific port rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.
list_options	The position in the list of ports from which SNAplus2 should begin to return data. Possible values are: AP_FIRST_IN_LIST Start at the first entry in the list. AP_LIST_INCLUSIVE Start at the entry specified by the *port_name* parameter. AP_LIST_FROM_NEXT Start at the entry immediately following the entry specified by the *port_name* parameter. For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.
fqcn_name	Fully qualified name of the CN on which the required port is defined, or the CN for which a list of ports is required. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

port_name Name of the port for which information is required, or the name to be used as an index into the list of ports. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cn_port_data.overlay_size

The size of the returned cn_port_data structure, and therefore the offset to the start of the next entry in the data buffer.

cn_port_data.fqcn_name

Fully qualified name of the CN. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a

network name of 1-8 A-string characters.

cn_port_data.port_name

Name of the port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

cn_port_data.tg_num

Transmission group number for the specified port.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_CN_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *fqcn_name* parameter was not valid.

AP_INVALID_PORT_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *port_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node is a LEN node, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node is a LEN node. This verb is valid only at an end node.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CONVERSATION

QUERY_CONVERSATION returns information about conversations using a particular local LU. This verb can be used to obtain information about a specific conversation or a range of conversations, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_conversation
{  AP_UINT16       opcode;                  /* verb operation code        */
   unsigned char   reserv2;                 /* reserved                   */
   unsigned char   format;                  /* reserved                   */
   AP_UINT16       primary_rc;              /* primary return code        */
   AP_UINT32       secondary_rc;            /* secondary return code      */
   unsigned char  *buf_ptr;                 /* pointer to buffer          */
   AP_UINT32       buf_size;                /* buffer size                */
   AP_UINT32       total_buf_size;          /* total buffer size required */
   AP_UINT16       num_entries;             /* number of entries          */
   AP_UINT16       total_num_entries;       /* total number of entries    */
   unsigned char   list_options;            /* listing options            */
   unsigned char   reserv3;                 /* reserved                   */
   unsigned char   lu_name[8];              /* LU Name                    */
   unsigned char   lu_alias[8];             /* LU Alias                   */
   AP_UINT32       conv_id;                 /* Conversation ID            */
   unsigned char   session_id[8];           /* Session ID                 */
   unsigned char   reserv4[12];             /* reserved                   */
 } QUERY_CONVERSATION;


typedef struct conv_summary
{  AP_UINT16      overlay_size;               /*overlay size              */
   AP_UINT32      conv_id;                    /* conversation ID          */
   unsigned char  local_tp_name[64];          /local TP name              */
   unsigned char  partner_tp_name[64];        /partner TP name           */
   unsigned char  tp_id[8];                   /TP ID                     */
   unsigned char  sess_id[8];                 /Session ID                */
   AP_UINT32      conv_start_time;            /Conversation start time   */
   AP_UINT32      bytes_sent;                 /Number of bytes sent      */
   AP_UINT32      bytes_received;             /Number of bytes received  */
   unsigned char  conv_state;                 /conversation state        */
```

```
  unsigned char  reserv1;                    /reserved                   */
} CONV_SUMMARY;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_CONVERSATION

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of conversations for which data should be returned. To request data for a specific conversation rather than a range, specify the value 1. To return as many entries as possible, sepcify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data.

Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of local LU and conversation ID.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry

specified by the combination of local LU and conversation ID.

The combination of the local LU (*lu_name* or *lu_alias* and conversation ID (*conv_id*) specified is used as an index into the list of sessions if the *list_options* parameter is set to AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

lu_name

LU name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To specify that the LU is identified by its alias rather than its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter. To specify the LU associated with the local CP (the default LU), set both lu_name and lu_alias to binary zeros.

lu_alias

Locally defined LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. This parameter is used only if lu_name is set to 8 binary zeros; it is ignored otherwise. To specify the LU associated with the local CP (the default LU), set both lu_name and lu_alias to binary zeros.

conv_id

Identifier of the conversation for which information is required, or the conversation ID to be used as an index into the list of conversations. The conversation ID was returned by the ALLOCATE verb in the invoking TP, or by the RECEIVE_ALLOCATE verb in the invoked TP.

This parameter is ignored if list_options is set to AP_FIRST_IN_LIST.

session_id

8-byte identifier of the session. To list only information

about conversations associated with a specific session, specify the session identifier. To obtain a complete list for all sessions, set this field to binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfull, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

REturned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than buf_size indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than num_entries indicates that not all the vailable entries were returned.

Each entry in the data buffer consists of the following parameters:

conv_summary.overlay_size

The size of the returned conv_summary structure, and therefore the offset to the start of the next entry in the data buffer.

When you r application needs to go through the returned buffer to find each conv_summary structure in turn, it must use this value to move to the correct offset for the next data structure, and must not use the C sizeof () operator. This is because the size of the

returned overlay may increase in furutre releases of SNAplus2; using the returned overlay size ensures that your application will continue to work with future releases.

conv_summary.conv_id

Conversation identifier. The conversation ID was returned by the ALLOCATE verb in the invoking TP, or by the RECEIVE_ALLOCATE verb in the invoked TP.

conv_summary.local_tp_name

The name of the local TP in the conversation.

conv_summary.partner_tp_name

The name of the partner TP in the conversation. This parameter is returned only if the conversation was started by the local TP; it is reserved if the conversation was started by the remote TP.

conv_summary.tp_id

The TP identifier of the conversation.

conv_summary.session_id

The session identifier of the session allocated to the conversation.

conv_summary.conv_start_time

The elapsed time in hundredths of seconds between the time when the SNAplusw node was started and the time when the conversation was started.

conv_summary.bytes_sent

The number of bytes that have been sent from the local TP to the partner TP since the start of the conversation.

conv_summary.bytes_received

The number of bytes that have been received from the partner TP by the local TP since the start of the conversation.

conv_summary.conv_state

The current state of the conversation. Values are:

AP_CONFIRM_STATE
AP_CONFIRM_DEALL_STATE
AP_CONFIRM_SEND_STATE
AP_END_CONV_STATE
AP_PEND_DEALL_STATE
AP_PEND_POST_STATE
AP_POST_ON_RECEIPT_STATE
AP_RECEIVE_STATE
AP_RESET_STATE
AP_SEND_STATE
AP_SEND_PENDING_STATE

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc — AP_PARAMETER_CHECK

secondary_rc — Possible values are:

AP_BAD_CONV_ID

The list_options parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied value, but the conv_id parameter was not valid

AP_INVALID_LU_ALIAS

The specivied lu_name parameter was not valid.

AP_INVALID_LU_NAME

The specified lu_name parameter was not valid.

AP_INVALID_LIST_OPTION

The list_options parameter was not set to a valid value.

Appendix A, “Common Return Codes,”lists further secondary return codes asociated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_COS

QUERY_COS returns route calculation information for a specific class of service (COS).

This verb can be used to obtain information about a specific COS or about multiple COSs, depending on the options used.

VCB Structure

```
typedef struct query_cos
{
   AP_UINT16        opcode;                /* verb operation code           */
   unsigned char    reserv2;               /* reserved                      */
   unsigned char    format;                /* reserved                      */
   AP_UINT16        primary_rc;            /* primary return code           */
   AP_UINT32        secondary_rc;          /* secondary return code         */
   unsigned char    *buf_ptr;              /* pointer to buffer             */
   AP_UINT32        buf_size;              /* buffer size                   */
   AP_UINT32        total_buf_size;        /* total buffer size required    */
   AP_UINT16        num_entries;           /* number of entries             */
   AP_UINT16        total_num_entries;     /* total number of entries       */
   unsigned char    list_options;          /* listing options               */
   unsigned char    reserv3;               /* reserved                      */
   unsigned char    cos_name[8];           /* cos name                      */
} QUERY_COS;


typedef struct cos_data
{
   AP_UINT16        overlay_size;          /* size of returned entry        */
   unsigned char    cos_name[8];           /* cos name                      */
   unsigned char    description[32];       /* resource description          */
   unsigned char    reserv1[16];           /* reserved                      */
   unsigned char    transmission_priority; /* transmission priority         */
   AP_UINT16        num_of_node_rows;      /* number of node rows           */
   AP_UINT16        num_of_tg_rows;        /* number of tg rows             */
   AP_UINT32        trees;                 /* number of tree caches for COS */
   AP_UINT32        calcs;                 /* number of route calculations  */
                                           /*  for this COS                 */
   AP_UINT32        rejs;                  /* number of route rejects for   */
                                           /*  COS                          */
   unsigned char    reserva[20];           /* reserved                      */
} COS_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_COS

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of COSs for which data should be returned. To request data for a specific COS rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of COSs from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *cos_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry

specified by the *cos_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

cos_name

Class of service name for which data is required, or the name to be used as an index into the list. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cos_data.overlay_size

The size of the returned cos_data structure, and therefore the offset to the start of the next entry in the data buffer.

cos_data.cos_name

Class of service name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

cos_data.description

A null-terminated text string describing the COS, as specified in the definition of the COS.

cos_data.transmission_priority

Transmission priority. Possible values are:

AP_LOW

AP_MEDIUM

AP_HIGH

AP_NETWORK (the highest priority)

cos_data.num_of_node_rows

Number of node rows defined for this COS.

cos_data.num_of_tg_rows

Number of TG rows defined for this COS.

cos_data.trees

Number of route tree caches built for this COS since the last initialization.

cos_data.calcs

Number of session activation requests (and therefore route calculations) specifying this class of service.

cos_data.rejs

Number of session activation requests that failed because there was no acceptable route from this node to the named destination through the network. A route is only acceptable if it is made up entirely of active TGs and nodes that can provide the specified class of

service.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_COS_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *cos_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_COS_NODE_ROW

QUERY_COS_NODE_ROW returns node row information for a specified class of service as previously defined by DEFINE_COS (or implicitly by the node for the SNA-defined COSs).

This verb can be used to obtain information about a specific COS node row, or about multiple COS node rows, depending on the options used.

VCB Structure

```
typedef struct query_cos_node_row
{
    AP_UINT16          opcode;              /* verb operation code          */
    unsigned char      reserv2;             /* reserved                     */
    unsigned char      format;              /* reserved                     */
    AP_UINT16          primary_rc;          /* primary return code          */
    AP_UINT32          secondary_rc;        /* secondary return code        */
    unsigned char      *buf_ptr;            /* pointer to buffer            */
    AP_UINT32          buf_size;            /* buffer size                  */
    AP_UINT32          total_buf_size;      /* total buffer size required   */
    AP_UINT16          num_entries;         /* number of entries            */
    AP_UINT16          total_num_entries;   /* total number of entries      */
    unsigned char      list_options;        /* listing options              */
    unsigned char      reserv3;             /* reserved                     */
    unsigned char      cos_name[8];         /* cos name                     */
    AP_UINT16          node_row_index;      /* node row index               */
} QUERY_COS_NODE_ROW;

typedef struct cos_node_row_data
{
    AP_UINT16          overlay_size;        /* size of returned entry       */
    unsigned char      cos_name[8];         /* cos name                     */
    AP_UINT16          node_row_index;      /* node row index               */
    COS_NODE_ROW       node_row;            /* cos node row information     */
} COS_NODE_ROW_DATA;

typedef struct cos_node_row
{
    COS_NODE_STATUS    minimum;             /* minimum                      */
    COS_NODE_STATUS    maximum;             /* maximum                      */
    unsigned char      weight;              /* weight                       */
```

```
    unsigned char      reserv1;              /* reserved                    */
} COS_NODE_ROW;

typedef struct cos_node_status
{
    unsigned char      rar;                  /* route additional resistance  */
    unsigned char      status;               /* node status                  */
    unsigned char      reserv1[2];           /* reserved                     */
} COS_NODE_STATUS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_COS_NODE_ROW

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of COS node rows for which data should be returned. To request data for a specific COS node row rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of COS node rows from which

SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *cos_name* and *node_row_index* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *cos_name* and *node_row_index* parameters.

The list is ordered by *cos_name*, and then by *node_row_index* for each COS. For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs".

cos_name

Class of service name for which node row information is required, or the name to be used as an index into the list. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

node_row_index

Node row number for which information is required, or the number to be used as an index into the list. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. Use QUERY_COS to determine the number of node rows associated with this COS.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cos_node_row_data.overlay_size

The size of the returned cos_node_row_data structure, and therefore the offset to the start of the next entry in the data buffer.

cos_node_row_data.cos_name

Class of service name. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

cos_node_row_data.node_row_index

Node row index.

cos_node_row_data.node_row.minimum.rar

Route additional resistance minimum, in the range 0-255.

cos_node_row_data.node_row.minimum.status

Specifies the minimum congestion status of the node. This parameter may be set to AP_UNCONGESTED, to any

one of the other values listed, or to two or more of the other values combined using a logical OR. Possible values are:

AP_UNCONGESTED

The number of ISR sessions is below the *isr_sessions_upper_threshold* value in the node's configuration.

AP_CONGESTED

The number of ISR sessions exceeds the threshold value.

AP_IRR_DEPLETED

The number of ISR sessions has reached the maximum specified for the node.

AP_ERR_DEPLETED

The number of endpoint sessions has reached the maximum specified.

AP_QUIESCING

A STOP_NODE of type AP_QUIESCE or AP_QUIESCE_ISR has been issued.

cos_node_row_data.node_row.maximum.rar

Route additional resistance maximum, in the range 0-255.

cos_node_row_data.node_row.maximum.status

Specifies the maximum congestion status of the node. This parameter may be set to AP_UNCONGESTED, to any one of the other values listed, or to two or more of the other values combined using a logical OR. Possible values are:

AP_UNCONGESTED

The number of ISR sessions is below the *isr_sessions_upper_threshold* value in the node's configuration.

AP_CONGESTED

The number of ISR sessions exceeds the threshold value.

AP_IRR_DEPLETED

The number of ISR sessions has reached the maximum specified for the node.

AP_ERR_DEPLETED

The number of endpoint sessions has reached the maximum specified.

AP_QUIESCING

A STOP_NODE of type AP_QUIESCE or AP_QUIESCE_ISR has been issued.

cos_node_row_data.node_row.weight

Weight associated with this node row.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_COS_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *cos_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_COS_TG_ROW

QUERY_COS_TG_ROW returns TG row information for a specified class of service as previously defined by DEFINE_COS (or implicitly by the node for the SNA-defined COSs).

This verb can be used to obtain information about a specific COS TG row, or about multiple COS TG rows, depending on the options used.

VCB Structure

```
typedef struct query_cos_tg_row
{
    AP_UINT16           opcode;             /* verb operation code         */
    unsigned char       reserv2;            /* reserved                    */
    unsigned char       format;             /* reserved                    */
    AP_UINT16           primary_rc;         /* primary return code         */
    AP_UINT32           secondary_rc;       /* secondary return code       */
    unsigned char       *buf_ptr;           /* pointer to buffer           */
    AP_UINT32           buf_size;           /* buffer size                 */
    AP_UINT32           total_buf_size;     /* total buffer size required  */
    AP_UINT16           num_entries;        /* number of entries           */
    AP_UINT16           total_num_entries;  /* total number of entries     */
    unsigned char       list_options;       /* listing options             */
    unsigned char       reserv3;            /* reserved                    */
    unsigned char       cos_name[8];        /* cos name                    */
    AP_UINT16           tg_row_index;       /* TG row index                */
} QUERY_COS_TG_ROW;

typedef struct cos_tg_row_data
{
    AP_UINT16           overlay_size;       /* size of returned entry      */
    unsigned char       cos_name[8];        /* cos name                    */
    AP_UINT16           tg_row_index;       /* TG row index                */
    COS_TG_ROW          tg_row;             /* TG row information          */
} COS_TG_ROW_DATA;

typedef struct cos_tg_row
{
    TG_DEFINED_CHARS    minimum;            /* minimum                     */
    TG_DEFINED_CHARS    maximum;            /* maximum                     */
    unsigned char       weight;             /* weight                      */
```

```
    unsigned char       reserv1;            /* reserved                    */
} COS_TG_ROW;


typedef struct tg_defined_chars
{
    unsigned char       effect_cap;         /* Effective capacity          */
    unsigned char       reserve1[5];        /* Reserved                    */
    unsigned char       connect_cost;       /* Connection Cost             */
    unsigned char       byte_cost;          /* Byte cost                   */
    unsigned char       reserve2;           /* Reserved                    */
    unsigned char       security;           /* Security                    */
    unsigned char       prop_delay;         /* Propagation delay           */
    unsigned char       modem_class;        /* reserved                    */
    unsigned char       user_def_parm_1;    /* User-defined parameter 1    */
    unsigned char       user_def_parm_2;    /* User-defined parameter 2    */
    unsigned char       user_def_parm_3;    /* User-defined parameter 3    */
} TG_DEFINED_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_COS_TG_ROW

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of COS TG rows for which data should be returned. To request data for a specific COS TG row rather than a range, specify the value 1. To

return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of COS TG rows from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *cos_name* and *tg_row_index* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *cos_name* and *tg_row_index* parameters.

The list is ordered by *cos_name*, and then by *tg_row_index* for each COS. For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

cos_name

Class of service name for which data is required, or the name to be used as an index into the list. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

tg_row_index

TG row number for which data is required, or the number to be used as an index into the list (the first row has an index of zero). This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cos_tg_row_data.overlay_size

The size of the returned cos_tg_row_data structure, and therefore the offset to the start of the next entry in the data buffer.

cos_tg_row_data.cos_name

Class of service name. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

cos_tg_row_data.tg_row_index

TG row index (the first row has an index of zero).

cos_tg_row_data.tg_row.minimum.effect_cap

Minimum limit for actual bits per second rate (line speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

cos_tg_row_data.tg_row.minimum.connect_cost

Minimum limit for cost per connect time; an integer value in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

cos_tg_row_data.tg_row.minimum.byte_cost

Minimum limit for cost per byte; an integer value in the range 0-255, where zero is the lowest cost per byte and 255 is the highest.

cos_tg_row_data.tg_row.minimum.security

Minimum level of security. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

cos_tg_row_data.tg_row.minimum.prop_delay

Minimum limits for propagation delay: the time that a signal takes to travel the length of the link. Possible values are:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN). If the verb was issued to a running node, this value will be returned if the DEFINE_COS specified either AP_PROP_DELAY_LAN or AP_PROP_DELAY_MINIMUM.

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

cos_tg_row_data.tg_row.minimum.user_def_parm_1 through *cos_tg_row_data.tg_row.minimum.user_def_parm_3*

Minimum values for user-defined parameters, which include other TG characteristics not covered by the above parameters. Each of these parameters is set to a value in the range 0-255.

cos_tg_row_data.tg_row.maximum.effect_cap

Maximum limit for actual bits per second rate (line

speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is eeeeemmm. Each unit of effective capacity is equal to 300 bits per second.

cos_tg_row_data.tg_row.maximum.connect_cost

Maximum limit for cost per connect time; an integer value in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

cos_tg_row_data.tg_row.maximum.byte_cost

Maximum limit for cost per byte; an integer value in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

cos_tg_row_data.tg_row.maximum.security

Maximum level of security. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

AP_SEC_MAXIMUM

Maximum security.

cos_tg_row_data.tg_row.maximum.prop_delay

Maximum limits for propagation delay: the time that a signal takes to travel the length of the link. Possible values are:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link). If the verb was issued to a running node, this value will be returned if the DEFINE_COS specified either AP_PROP_DELAY_SATELLITE or AP_PROP_DELAY_MAXIMUM.

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

cos_tg_row_data.tg_row.maximum.user_def_parm_1 through *cos_tg_row_data.tg_row.maximum.user_def_parm_3*

Maximum values for user-defined parameters, which include other TG characteristics not covered by the above parameters. Each of these parameters is set to a value in the range 0-255.

cos_tg_row_data.tg_row.weight

Weight associated with this TG row.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_COS_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *cos_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CPIC_SIDE_INFO

This verb returns the side information entry for a given symbolic destination name, or for multiple symbolic destination names, depending on the options used.

Note the difference between this verb and the CPI-C function `Extract_CPIC_Side_Information()`. This verb queries a configuration file, so that it returns the default information used by all SNAplus2 CPI-C applications. The CPI-C function queries the application's own copy in memory of the side information table, which the application may have modified using the other CPI-C side information functions.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct query_cpic_side_info
{
   AP_UINT16            opcode;              /* verb operation code         */
   unsigned char        reserv2;             /* reserved                    */
   unsigned char        format;              /* reserved                    */
   AP_UINT16            primary_rc;          /* primary return code         */
   AP_UINT32            secondary_rc;        /* secondary return code       */
   unsigned char        *buf_ptr;            /* pointer to buffer           */
   AP_UINT32            buf_size;            /* buffer size                 */
   AP_UINT32            total_buf_size;      /* total buffer size required  */
   AP_UINT16            num_entries;         /* number of entries           */
   AP_UINT16            total_num_entries;   /* total number of entries     */
   unsigned char        list_options;        /* listing options             */
   unsigned char        reserv3;             /* reserved                    */
   unsigned char        sym_dest_name[8];    /* Symbolic destination name   */
} QUERY_CPIC_SIDE_INFO;

typedef struct cpic_side_info_data
{
   AP_UINT16                overlay_size;     /* size of returned entry     */
   unsigned char            sym_dest_name[8]; /* Symbolic destination name  */
   unsigned char            reserv1[2];       /* reserved                   */
   CPIC_SIDE_INFO_DEF_DATA  def_data;
} CPIC_SIDE_INFO_DATA;
```

```
typedef struct cpic_side_info_def_data
{
   unsigned char             description[32];  /* resource description       */
   unsigned char             reserv1[16];      /* reserved                  */
   CPIC_SIDE_INFO            side_info;        /* CPIC side info            */
   unsigned char             reserv2[24];      /* reserved                  */
} CPIC_SIDE_INFO_DEF_DATA;

typedef struct cpic_side_info
{
   unsigned char   partner_lu_name[17];        /* Fully qualified partner    */
                                               /*  LU name                   */
   unsigned char   reserved[3];                /* Reserved                  */
   AP_UINT32       tp_name_type;               /* TP name type              */
   unsigned char   tp_name[64];                /* TP name                   */
   unsigned char   mode_name[8];               /* Mode name                 */
   AP_UINT32       conversation_security_type; /* Conversation security      */
                                               /*  type                     */
   unsigned char   security_user_id[10];       /* User ID                   */
   unsigned char   security_password[10];      /* Password                  */
   unsigned char   lu_alias[8];                /* LU alias                  */
} CPIC_SIDE_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_CPIC_SIDE_INFO

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of symbolic destination names for which data should be returned. To request data for a specific symbolic destination name rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of symbolic destination names from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *sym_dest_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *sym_dest_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

sym_dest_name

Symbolic destination name for which data is required, or the name to be used as an index into the list. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an ASCII string, consisting of uppercase A-Z and numerals 0-9, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

cpic_side_info_data.overlay_size

The size of the returned cpic_side_info_data structure, and therefore the offset to the start of the next entry in the data buffer.

cpic_side_info_data.sym_dest_name

Symbolic destination name for the returned side information entry.

cpic_side_info_data.def_data.description

A null-terminated text string describing the side information entry, as specified in the definition of the side information entry.

cpic_side_info_data.def_data.side_info.partner_lu_name

Fully qualified name of the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string

characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

cpic_side_info_data.def_data.side_info.tp_name_type

The type of the target TP (the valid characters for a TP name are determined by the TP type). Possible values are:

XC_APPLICATION_TP

Application TP. All characters in the TP name must be valid ASCII characters.

XC_SNA_SERVICE_TP

Service TP. All characters, except the first, in the TP name must be valid ASCII characters. The first character must be a hexadecimal digit in the range 0x01-0x3F, excluding 0x0E and 0x0F.

cpic_side_info_data.def_data.side_info.tp_name

TP name of the target TP. This is a 64-byte ASCII character string, right-padded with spaces.

cpic_side_info_data.def_data.side_info.mode_name

Name of the mode used to access the target TP. This is an 8-byte ASCII character string, right-padded with spaces.

cpic_side_info_data.def_data.side_info.conversation_security_ type

Specifies whether the target TP uses conversation security. Possible values are:

XC_SECURITY_NONE

The target TP does not use conversation security.

XC_SECURITY_PROGRAM

The target TP uses conversation security. The *security_user_id* and *security_password* parameters specified below will be used to access the target TP.

XC_SECURITY_PROGRAM_STRONG

As for XC_SECURITY_PROGRAM, except that the local node must not send the password across the network in clear text format. (This value is included for compatibility with IBM CPI-C implementations. The SNAplus2 node cannot provide the appropriate restrictions on sending the password; if a CPI-C application attempts to issue the Allocate call with this value set, the call will fail with a return code indicating that the requested security type is not supported.)

XC_SECURITY_SAME

The target TP uses conversation security, and can accept an "already verified" indicator from the local TP. (This indicates that the local TP was itself invoked by another TP, and has verified the security user ID and password supplied by this TP.) The *security_user_id* parameter specified below will be used to access the target TP; no password is required.

cpic_side_info_data.def_data.side_info.security_user_id

User ID used to access the partner TP. This parameter is not used if the *conversation_security_type* parameter is set to XC_SECURITY_NONE.

cpic_side_info_data.def_data.side_info.security_password

Password used to access the partner TP. This parameter is used only if the *conversation_security_type* parameter is set to XC_SECURITY_PROGRAM or XC_SECURITY_PROGRAM_STRONG.

cpic_side_info_data.def_data.side_info.lu_alias

The alias of the local LU used to communicate with the target TP. This alias is a character string using any locally displayable characters.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_SYM_DEST_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *sym_dest_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_CS_TRACE

This verb returns information about the current tracing options for data sent between computers on the SNAplus2 LAN. For more information about tracing options, see the *HP-UX SNAplus2 Administration Guide*.

This verb may be issued to a running node, or to a HP-UX client computer on which the SNAplus2 software is running. To obtain a target handle for the client in order to issue this verb, use the CONNECT_NODE verb without specifying a node name; the NOF application must be running on the client.

On Windows clients, client-server tracing is controlled by options in the `sna.ini` file. For more information, see the *HP-UX SNAplus2 Administration Guide*.

VCB Structure

```
typedef struct query_cs_trace
{
   AP_UINT16        opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   AP_UINT16        primary_rc;       /* primary return code               */
   AP_UINT32        secondary_rc;     /* secondary return code             */
   unsigned char    dest_sys[64];     /* node to which messages are traced  */
   AP_UINT16        trace_flags;      /* trace flags                       */
   AP_UINT16        trace_direction;  /* direction (send/rcv/both) to trace */
} QUERY_CS_TRACE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_CS_TRACE

dest_sys The server name for which tracing options are being queried. This is an ASCII string, padded on the right with spaces if the name is shorter than 64 characters.

To query tracing options on messages flowing between the computer to which this verb is issued (identified by

the *target_handle* parameter on the NOF API call) and one other server on the LAN, specify the name of the other server here.

To query the default tracing options (set by a SET_CS_TRACE verb with no destination system name specified), set this parameter to 64 ASCII space characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

trace_flags

The types of tracing currently active. For more information about these trace types, see "SET_CS_TRACE".

If no tracing is active, or if tracing of all types is active, this is one of the following values:

AP_NO_TRACE

No tracing.

AP_ALL_TRACE

Tracing of all types.

If tracing is being used on specific interfaces, this parameter is set to one or more values from the list below, combined using a logical OR operation.

AP_CS_ADMIN_MSG

Internal messages relating to client-server topology

AP_CS_DATAGRAM

Datagram messages

AP_CS_DATA

Data messages

trace_direction

Specifies the direction or directions in which tracing is active. This parameter is not used if *trace_flags* is set to AP_NO_TRACE. Possible values are:

AP_CS_SEND

Messages flowing from the target computer to the computer defined by *dest_sys* are traced.

AP_CS_RECEIVE

Messages flowing from the computer defined by *dest_sys* to the target computer are traced.

AP_CS_BOTH

Messages flowing in both directions are traced.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_NAME_NOT_FOUND

The server specified by the *dest_sys* parameter did not exist or was not started.

AP_LOCAL_SYSTEM

The server specified by the *dest_sys* parameter is the same as the target node to which this verb was issued.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of

primary and secondary return codes that are common to all NOF verbs.

QUERY_DEFAULT_PU

QUERY_DEFAULT_PU allows the user to query the default PU (defined using DEFINE_DEFAULT_PU).

VCB Structure

```
typedef struct query_default_pu
{
   AP_UINT16        opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   AP_UINT16        primary_rc;       /* primary return code               */
   AP_UINT32        secondary_rc;     /* secondary return code             */
   unsigned char    def_pu_name[8];   /* default PU name                   */
   unsigned char    description[32];  /* resource description              */
   unsigned char    reserv1[16];      /* reserved                          */
   unsigned char    def_pu_sess[8];   /* PU name of active default session */
   unsigned char    reserv3[16];      /* reserved                          */
} QUERY_DEFAULT_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DEFAULT_PU

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

def_pu_name

Name of the PU specified on the most recent DEFINE_DEFAULT_PU verb. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the

right with EBCDIC spaces. If this field is set to all binary zeros, this indicates that no DEFINE_DEFAULT_PU verb has been issued or that the default PU has been deleted by issuing a DEFINE_DEFAULT_PU verb with the *pu_name* parameter specified as all zeros.

description

A null-terminated text string describing the default PU, as specified in the definition of the default PU.

def_pu_sess

Name of the PU associated with the currently active default PU session.

This parameter normally contains the same value as the *def_pu_name* field. However, if a default PU has been defined, but the session associated with it is not active, SNAplus2 continues to use the session associated with the previous default PU until the session associated with the defined default PU becomes active. In this case, this parameter specifies the name of the previous default PU, and is different from the *def_pu_name* field.

If there are no active PU sessions, this field will be set to all binary zeros.

Returned Parameters: Node Not Started

If the verb does not execute because the node has not yet been started, SNAplus2 returns the following parameters:

primary_rc AP_NODE_NOT_STARTED

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DEFAULTS

QUERY_DEFAULTS allows the user to query the default parameters defined for the node (defined using DEFINE_DEFAULTS).

VCB Structure

```
typedef struct query_defaults
{
     AP_UINT16         opcode;             /* verb operation code            */
     unsigned char     reserv2;            /* reserved                       */
     unsigned char     format;             /* reserved                       */
     AP_UINT16         primary_rc;         /* primary return code            */
     AP_UINT32         secondary_rc;       /* secondary return code          */
     DEFAULT_CHARS     default_chars;      /* default parameters             */
} QUERY_DEFAULTS;

typedef struct default_chars
{
     unsigned char     description[32];    /* resource description           */
     unsigned char     reserv2[16];        /* reserved                       */
     unsigned char     mode_name[8];       /* default mode name              */
     unsigned char     implicit_plu_forbidden; /* disallow implicit PLUs?    */
     unsigned char     specific_security_codes;/*generic security sensecodes?*/
     AP_UINT16         limited_timeout;    /* timeout for limited sessions   */
     unsigned char     reserv[244];        /* reserved                       */
} DEFAULT_CHARS;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_DEFAULTS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

default_chars.description

A null-terminated text string describing the default parameters, as specified in DEFINE_DEFAULTS.

default_chars.mode_name

Name of the default mode. If an application specifies an unrecognized mode name when attempting to start a session, the parameters from this mode will be used as a default definition for the unrecognized mode.

The mode name is an 8-byte type-A EBCDIC string. If no default mode name has been specified using the DEFINE_DEFAULTS verb, this parameter is set to 8 binary zeros.

default_chars.implicit_plu_forbidden

Indicates whether SNAplus2 puts implicit definitions in place for unknown partner LUs. Possible values are:

AP_YES

SNAplus2 puts implicit definitions in place for unknown partner LUs.

AP_NO

SNAplus2 does not put implicit definitions in place for unknown partner LUs.

default_chars.specific_security_codes

Indicates whether SNAplus2 uses specific sense codes on a security authentication or authorization failure. Specific sense codes are only returned to those partner LUs which have reported support for them on the session. Possible values are:

AP_YES

SNAplus2 uses specific sense codes.

AP_NO

SNAplus2 does not use specific sense codes.

default_chars.limited_timeout

Specifies the timeout after which free limited-resource

conwinner sessions are deactivated. The range is 0-65,535 seconds.

Returned Parameters: Node Not Started

If the verb does not execute because the node has not yet been started, SNAplus2 returns the following parameter:

primary_rc AP_NODE_NOT_STARTED

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DIRECTORY_ENTRY

QUERY_DIRECTORY_ENTRY returns information about resources in the directory database. It can return either summary or detailed information, about a specific resource or multiple resources, depending on the options used.

If the verb is issued to a running node, it returns information both on resources that have been defined explicitly (using DEFINE_DIRECTORY_ENTRY, or DEFINE_ADJACENT_LEN_NODE) and on resources that have been located dynamically. If the node is not running, only explicitly defined entries are returned.

When the verb is issued to an end node, the directory contains only information about the end node and its resources, and not about other nodes. The first entry returned is for the end node itself, followed by its LUs. (No entry is returned for the end node's network node server.)

VCB Structure

```
typedef struct query_directory_entry
{
   AP_UINT16         opcode;              /* verb operation code           */
   unsigned char     reserv2;             /* reserved                      */
   unsigned char     format;              /* reserved                      */
   AP_UINT16         primary_rc;          /* primary return code           */
   AP_UINT32         secondary_rc;        /* secondary return code         */
   unsigned char     *buf_ptr;            /* pointer to buffer             */
   AP_UINT32         buf_size;            /* buffer size                   */
   AP_UINT32         total_buf_size;      /* total buffer size required    */
   AP_UINT16         num_entries;         /* number of entries             */
   AP_UINT16         total_num_entries;   /* total number of entries       */
   unsigned char     list_options;        /* listing options               */
   unsigned char     reserv3;             /* reserved                      */
   unsigned char     resource_name[17];   /* network qualified resource    */
                                          /* name                          */
   unsigned char     reserv4;             /* reserved                      */
   AP_UINT16         resource_type;       /* Resource type                 */
   unsigned char     parent_name[17];     /* parent name filter            */
   unsigned char     reserv5;             /* reserved                      */
   AP_UINT16         parent_type;         /* parent type                   */
} QUERY_DIRECTORY_ENTRY;
```

```
typedef struct directory_entry_summary
{
   AP_UINT16         overlay_size;          /* size of this entry            */
   unsigned char     resource_name[17];     /* network qualified resource    */
                                            /* name                          */
   unsigned char     reserve1;              /* reserved                      */
   AP_UINT16         resource_type;         /* Resource type                 */
   description       description;           /* resource description          */
} DIRECTORY_ENTRY_SUMMARY;

typedef struct directory_entry_detail
{
   AP_UINT16         overlay_size;          /* size of this entry            */
   unsigned char     resource_name[17]      /* network qualified res name    */
   unsigned char     reserv1a;              /* reserved                      */
   AP_UINT16         resource_type;         /* Resource type                 */
   description       description;           /* resource description          */
   unsigned char     parent_name[17];       /* Network qualified parent name */
   unsigned char     reserv1b;              /* reserved                      */
   AP_UINT16         parent_type;           /* Parent resource type          */
   unsigned char     entry_type;            /* Type of the directory entry   */
   unsigned char     location;              /* Resource location             */
   unsigned char     reserva[20]            /* reserved                      */
{ DIRECTORY_ENTRY_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DIRECTORY_ENTRY

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of resources for which data should be returned. To request data for a specific resource rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *parent_name*, *resource_name*, and *resource_type* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *parent_name*, *resource_name*, and *resource_type* parameters.

The list is ordered by *parent_name*, then by *resource_name*, and lastly by *resource_type*. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

resource_name

Fully qualified name of the resource for which information is required, or the name to be used as an index into the list of resources. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

resource_type

Type of resource for which information is required. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. Possible values are:

AP_ENCP_RESOURCE

End node or LEN node

AP_NNCP_RESOURCE

Network node

AP_LU_RESOURCE

LU

parent_name

Fully qualified resource name of the parent resource; for an LU the parent resource is the owning Control Point, and for an end node or LEN node it is the network node server. To return only entries belonging to the specified parent, set this parameter to the name of the parent resource and *parent_type* to the parent's resource type; to return all entries, set both parameters to binary zeros.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

parent_type

Resource type of the parent resource. To return only entries belonging to the specified parent, set this parameter to the type of the parent resource; to return all entries, set this parameter to zero. Possible values are:

AP_ENCP_RESOURCE

End node (for an LU resource owned by an end node)

AP_NNCP_RESOURCE

Network node (for an LU resource owned by a network node, or for an EN or LEN resource)

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

directory_entry_summary.overlay_size

directory_entry_summaryThe size of the returned

structure, and therefore the offset to the start of the next entry in the data buffer.

directory_entry_summary.resource_name

Fully qualified name of the resource. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters

directory_entry_summary.resource_type

Type of the resource. This is one of the following:

AP_ENCP_RESOURCE

End node or LEN node

AP_NNCP_RESOURCE

Network node

AP_LU_RESOURCE

LU

directory_entry_summary.description

A null-terminated text string describing the directory entry, as specified in the definition of the directory entry.

directory_entry_detail.overlay_size

The size of the returned directory_entry_detail structure, and therefore the offset to the start of the next entry in the data buffer.

directory_entry_detail.resource_name

Fully qualified name of the resource. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_entry_detail.resource_type

Type of the resource. This is one of the following:

AP_ENCP_RESOURCE

End node or LEN node

AP_NNCP_RESOURCE

Network node

AP_LU_RESOURCE

LU

directory_entry_detail.description

A null-terminated text string describing the directory entry, as specified in the definition of the directory entry.

directory_entry_detail.parent_name

Fully qualified resource name of the parent resource; for an LU the parent resource is the owning Control Point, and for an end node or LEN node it is the network node server. This parameter is not used for a network node resource.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_entry_detail.parent_type

Resource type of the parent resource. For a network node resource, this parameter is not used. Otherwise, it is one of the following:

AP_ENCP_RESOURCE

End node (for an LU resource owned by an end node)

AP_NNCP_RESOURCE

Network node (for an LU resource owned by a network node, or for an EN or LEN resource)

directory_entry_detail.entry_type

Specifies the type of the directory entry. This is one of the following:

AP_HOME

Local resource.

AP_CACHE

Cached entry.

directory_entry_detail.location

Specifies the location of the resource. This is one of the following.

AP_LOCAL

The resource is at the local node.

AP_DOMAIN

The resource belongs to an attached end node.

AP_CROSS_DOMAIN

The resource is not within the domain of the local node.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_RES_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *resource_name* parameter was not valid.

AP_INVALID_RES_TYPE

The *resouce_type* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DIRECTORY_LU

QUERY_DIRECTORY_LU returns a list of LUs from the directory database. It can be used to obtain information about a specific LU, or about multiple LUs, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_directory_lu
{
   AP_UINT16         opcode;              /* verb operation code           */
   unsigned char     reserv2;             /* reserved                      */
   unsigned char     format;              /* reserved                      */
   AP_UINT16         primary_rc;          /* primary return code           */
   AP_UINT32         secondary_rc;        /* secondary return code         */
   unsigned char     *buf_ptr;            /* pointer to buffer             */
   AP_UINT32         buf_size;            /* buffer size                   */
   AP_UINT32         total_buf_size;      /* total buffer size required    */
   AP_UINT16         num_entries;         /* number of entries             */
   AP_UINT16         total_num_entries;   /* total number of entries       */
   unsigned char     list_options;        /* listing options               */
   unsigned char     reserv3;             /* reserved                      */
   unsigned char     lu_name[17];         /* network qualified lu name     */
} QUERY_DIRECTORY_LU;


typedef struct directory_lu_summary
{
   AP_UINT16         overlay_size;        /* size of returned entry        */
   unsigned char     lu_name[17];         /* network qualified lu name     */
   unsigned char     description[32];     /* resource description          */
   unsigned char     reserv1[16];         /* reserved                      */
} DIRECTORY_LU_SUMMARY;


typedef struct directory_lu_detail
{
   AP_UINT16         overlay_size;        /* size of returned entry        */
   unsigned char     lu_name[17];         /* network qualified lu name     */
   unsigned char     description[32];     /* resource description          */
   unsigned char     reserv1[16];         /* reserved                      */
   unsigned char     server_name[17];     /* network qualified server name */
```

```
   unsigned char     lu_owner_name[17];    /* network qualified lu owner name*/
   unsigned char     location;             /* Resource location              */
   unsigned char     entry_type;           /* Type of the directory entry    */
   unsigned char     wild_card;            /* type of wildcard entry         */
   unsigned char     reserva[20];          /* reserved                       */
} DIRECTORY_LU_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DIRECTORY_LU

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *lu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *lu_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

lu_name

Fully qualified name of the LU for which information is required, or the name to be used as an index into the list of LUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied

buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

directory_lu_summary.overlay_size

The size of the returned directory_lu_summary structure, and therefore the offset to the start of the next entry in the data buffer.

directory_lu_summary.lu_name

Fully qualified name of the LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_lu_summary.description

A null-terminated text string describing the directory entry, as specified in the definition of the directory entry.

directory_lu_detail.overlay_size

The size of the returned directory_lu_detail structure, and therefore the offset to the start of the next entry in the data buffer.

directory_lu_detail.lu_name

Fully qualified name of the LU. The name is a 17-byte

EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_lu_detail.description

A null-terminated text string describing the directory entry, as specified in the definition of the directory entry.

directory_lu_detail.server_name

Fully qualified name of the node that serves the LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_lu_detail.lu_owner_name

Fully qualified name of the node that owns the LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

directory_lu_detail.location

Specifies the location of the resource. This is one of the following.

AP_LOCAL

The resource is at the local node.

AP_DOMAIN

The resource belongs to an attached end node.

AP_CROSS_DOMAIN

The resource is not within the domain of the local node.

directory_lu_detail.entry_type

Specifies the type of the resource. This is one of the following:

AP_HOME

Local resource.

AP_CACHE

Cached entry.

directory_lu_detail.wild_card

Specifies whether the LU entry is for an explicit name, or for a wildcard value that will match a range of names. This is one of the following:

AP_EXPLICIT

The entry is an explicit LU name.

AP_FULL_WILDCARD

The entry is a full wildcard value that will match any LU name.

AP_PARTIAL_WILDCARD

The entry is a partial wildcard; the nonblank characters in the name will be used to match against an LU name.

AP_OTHER

Unknown type of LU entry.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DIRECTORY_STATS

QUERY_DIRECTORY_STATS returns directory database statistics, which can be used to gauge the level of network locate traffic.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_directory_stats
{
   AP_UINT16        opcode;                  /* verb operation code         */
   unsigned char    reserv2;                 /* reserved                    */
   unsigned char    format;                  /* reserved                    */
   AP_UINT16        primary_rc;              /* primary return code         */
   AP_UINT32        secondary_rc;            /* secondary return code       */
   AP_UINT32        max_caches;              /* maximum number of cache     */
                                             /*  entries                    */
   AP_UINT32        cur_caches;              /* cache entry count           */
   AP_UINT32        cur_home_entries;        /* home entry count            */
   AP_UINT32        cur_reg_entries;         /* registered entry count      */
   AP_UINT32        cur_directory_entries;   /* current number of directory */
                                             /*  entries                    */
   AP_UINT32        cache_hits;              /* count of cache finds        */
   AP_UINT32        cache_misses;            /* count of resources found    */
                                             /*  by broadcast search        */
                                             /*  (not in cache)             */
   AP_UINT32        in_locates;              /* locates in                  */
   AP_UINT32        in_bcast_locates;        /* broadcast locates in        */
   AP_UINT32        out_locates;             /* locates out                 */
   AP_UINT32        out_bcast_locates;       /* broadcast locates out       */
   AP_UINT32        not_found_locates;       /* unsuccessful locates        */
   AP_UINT32        not_found_bcast_locates; /* unsuccessful broadcast      */
                                             /*  locates                    */
   AP_UINT32        locates_outstanding;     /* total outstanding locates   */
   unsigned char    reserva[20];             /* reserved                    */
} QUERY_DIRECTORY_STATS;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_DIRECTORY_STATS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

max_caches

This parameter is reserved.

cur_caches

This parameter is reserved.

cur_home_entries

Current number of home entries.

cur_reg_entries

Current number of registered entries.

cur_directory_entries

Total number of entries currently in the directory.

cache_hits

This parameter is reserved.

cache_misses

This parameter is reserved.

in_locates

Number of directed locates received.

in_bcast_locates

This parameter is reserved.

out_locates

Number of directed locates sent.

out_bcast_locates

This parameter is reserved.

not_found_locates

Number of directed locates returned “not found”.

not_found_bcast_locates

This parameter is reserved.

locates_outstanding

Current number of outstanding locates.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLC

QUERY_DLC returns information about DLCs. This information is structured as “determined data” (data gathered dynamically during execution) and “defined data” (data supplied on DEFINE_DLC).

This verb can be used to obtain either summary or detailed information, about a specific DLC or about multiple DLCs, depending on the options used.

VCB Structure

```
typedef struct query_dlc
{
   AP_UINT16       opcode;              /* verb operation code              */
   unsigned char   reserv2;             /* reserved                         */
   unsigned char   format;              /* reserved                         */
   AP_UINT16       primary_rc;          /* primary return code              */
   AP_UINT32       secondary_rc;        /* secondary return code            */
   unsigned char   *buf_ptr;            /* pointer to buffer                */
   AP_UINT32       buf_size;            /* buffer size                      */
   AP_UINT32       total_buf_size;      /* total buffer size required       */
   AP_UINT16       num_entries;         /* number of entries                */
   AP_UINT16       total_num_entries;   /* total number of entries          */
   unsigned char   list_options;        /* listing options                  */
   unsigned char   reserv3;             /* reserved                         */
   unsigned char   dlc_name[8];         /* name of DLC                      */
} QUERY_DLC;


typedef struct dlc_summary
{
   AP_UINT16       overlay_size;        /* size of returned entry           */
   unsigned char   dlc_name[8];         /* name of DLC                      */
   unsigned char   description[32];     /* resource description             */
   unsigned char   reserv1[16];         /* reserved                         */
   unsigned char   state;               /* State of the DLC                 */
   unsigned char   dlc_type;            /* DLC type                         */
}DLC_SUMMARY;


typedef struct dlc_detail
{
   AP_UINT16       overlay_size;        /* size of returned entry           */
```

```
   unsigned char   dlc_name[8];          /* name of DLC                    */
   unsigned char   reserv2[2];           /* reserved                       */
   DLC_DET_DATA    det_data;             /* Determined data                */
   DLC_DEF_DATA    def_data;             /* Defined data                   */
} DLC_DETAIL;

typedef struct dlc_det_data
{
   unsigned char   state;                /* State of the DLC               */
   unsigned char   reserv3[3];           /* reserved                       */
   unsigned char   reserva[20];          /* reserved                       */
}DLC_DET_DATA;

typedef struct dlc_def_data
{
   unsigned char   description[32];      /* resource description           */
   unsigned char   initially_active;     /* is DLC initially active?       */
   unsigned char   reserv1[15];          /* reserved                       */
   unsigned char   dlc_type;             /* DLC type                       */
   unsigned char   neg_ls_supp;          /* negotiable link station support  */
   unsigned char   port_types;           /* port types supported by DLC type */
   unsigned char   reserv3[11];          /* reserved                       */
   AP_UINT16       dlc_spec_data_len;    /* Length of DLC specific data    */
} DLC_DEF_DATA;
```

For more details of the DLC-specific data, see "DEFINE_DLC". The data structure for this data follows the `dlc_def_data` structure, but is padded to start on a 4-byte boundary.

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLC

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of DLCs for which data should be returned. To request data for a specific DLC rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *dlc_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *dlc_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

dlc_name

DLC name for which information is required, or the name to be used as an index into the list of DLCs. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

dlc_summary.overlay_size

The size of the returned dlc_summary structure, and therefore the offset to the start of the next entry in the data buffer.

dlc_summary.dlc_name

DLC name. The name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter

than 8 characters.

dlc_summary.description

A null-terminated text string describing the DLC, as specified in the definition of the DLC.

dlc_summary.state

State of the DLC. This is one of the following:

AP_ACTIVE

The DLC is active.

AP_NOT_ACTIVE

The DLC is not active.

AP_PENDING_INACTIVE

STOP_DLC is in progress.

dlc_summary.dlc_type

Type of DLC. This is one of the following:

AP_SDLC

SDLC

AP_X25

QLLC

AP_TR

Token Ring

AP_ETHERNET

Ethernet

AP_FDDI

FDDI

dlc_detail.overlay_size

The size of the returned dlc_detail structure, and therefore the offset to the start of the next entry in the data buffer.

dlc_detail.dlc_name

DLC name. The name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

dlc_detail.det_data.state

State of the DLC. This is one of the following:

AP_ACTIVE

The DLC is active.

AP_NOT_ACTIVE

The DLC is not active.

AP_PENDING_INACTIVE

STOP_DLC is in progress.

dlc_detail.def_data.description

A null-terminated text string describing the DLC, as specified in the definition of the DLC.

dlc_detail.def_data.initially_active

Specifies whether this DLC is automatically started when the node is started. Possible values are:

AP_YES

The DLC is automatically started when the node is started.

AP_NO

The DLC is not automatically started; it must be started manually.

dlc_detail.def_data.dlc_type

Type of DLC. This is one of the following:

AP_SDLC	SDLC
AP_X25	QLLC
AP_TR	Token Ring
AP_ETHERNET	Ethernet
AP_FDDI	FDDI

dlc_detail.def_data.neg_ls_supp

Specifies whether the DLC supports negotiable link stations. Possible values are:

AP_YES

Link stations using this DLC may be negotiable.

AP_NO

Link stations using this DLC must be defined as either primary or secondary; negotiable link stations are not supported.

dlc_detail.def_data.port_types

If *dlc_type* is set to AP_ETHERNET /AP_FDDI, this parameter will be set to AP_PORT_SATF. For other DLC types, this parameter is reserved.

dlc_detail.def_data.dlc_spec_data_len

Unpadded length, in bytes, of data specific to the type of DLC. The data structure for this data follows the def_data structure, but is padded to start on a 4-byte boundary. For more details of the DLC-specific data, see “DEFINE_DLC”.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DLC_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *dlc_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLC_TRACE

QUERY_DLC_TRACE returns information about DLC line tracing, which was set up using ADD_DLC_TRACE verbs.

This verb can be used to obtain information about tracing on all resources, on a specific resource type, or on a specific resource, depending on the options used.

VCB Structure

```
typedef struct query_dlc_trace
{
    AP_UINT16          opcode;              /* Verb operation code          */
    unsigned char      reserv2;             /* reserved                     */
    unsigned char      format;              /* reserved                     */
    AP_UINT16          primary_rc;          /* Primary return code          */
    AP_UINT32          secondary_rc;        /* Secondary return code        */
    unsigned char      *buf_ptr;            /* pointer to buffer            */
    AP_UINT32          buf_size;            /* buffer size                  */
    AP_UINT32          total_buf_size;      /* total buffer size required   */
    AP_UINT16          num_entries;         /* number of entries            */
    AP_UINT16          total_num_entries;   /* total number of entries      */
    unsigned char      list_options;        /* listing options              */
    unsigned char      list_type;           /* type of listing required     */
    DLC_TRACE_FILTER   filter;              /* resource to start at         */
} QUERY_DLC_TRACE;


typedef struct dlc_trace_data
{
    AP_UINT16          overlay_size;        /* size of returned entry       */
    DLC_TRACE_FILTER   filter;              /* DLC trace filter information */
} DLC_TRACE_DATA;


typedef struct dlc_trace_filter
{
    unsigned char      resource_type;       /* type of resource             */
    unsigned char      resource_name[8];    /* name of resource             */
    SNA_LFSID          lfsid;               /* session identifier           */
    unsigned char      message_type         /* type of messages             */
} DLC_TRACE_FILTER;
```

```
typedef struct sna_lfsid
{
      union
      {
           AP_UINT16            session_id;
           struct
           {
              unsigned char     sidh;
              unsigned char     sidl;
           } s;
      } uu;
      AP_UINT16                 odai;
} SNA_LFSID;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLC_TRACE

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of DLC_TRACE entries for which data should be returned. To request data for a specific entry rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of DLC_TRACE entries from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the `filter` structure.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the `filter` structure.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

list_type

The type of resource for which to list tracing options. Possible values are:

AP_ALL_DLC_TRACES

List all specified tracing options (for any resource type).

AP_ALL_RESOURCES

List the tracing options specified for all resources (defined using ADD_DLC_TRACE with a resource type of AP_ALL_RESOURCES).

AP_DLC

List tracing options for DLC resources.

AP_PORT

List tracing options for port resources for which all LSs are traced.

AP_LS

List tracing options for LS resources.

AP_PORT_DEFINED_LS

List tracing options for port resources for which only defined LSs (not implicit LSs) are traced.

AP_PORT_IMPLICIT_LS

List tracing options for port resources for which only implicit LSs (not defined LSs) are traced.

filter.resource_type

Specifies the resource type of the entry to be returned, or the entry to be used as an index into the list. This parameter is used only if *list_type* is set to AP_ALL_DLC_TRACES and *list_options* is not set to AP_FIRST_IN_LIST. Possible values are:

AP_ALL_RESOURCES

The required entry specifies the options used for tracing all DLCs, ports, and LSs.

AP_DLC

The required entry specifies tracing options for the DLC named in *resource_name*, and for all ports and LSs that use this DLC.

AP_PORT

The required entry specifies tracing options for the port named in *resource_name*, and for all LSs that use this port.

AP_LS

The required entry specifies tracing options for the LS named in *resource_name*.

AP_PORT_DEFINED_LS

The required entry specifies tracing options for the port named in *resource_name*, and for all defined LSs (but not implicit LSs) that use this port.

AP_PORT_IMPLICIT_LS

The required entry specifies tracing options for the port named in *resource_name*, and for all implicit LSs (but not defined LSs) that use this port.

filter.resource_name

The name of the entry to be returned, or the entry to be used as an index into the list. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST, or if *resource_type* is set to AP_ALL_RESOURCES.

filter.lfsid

The Local Form Session Identifier for a session on the specified LS. This is only valid for *resource_type* AP_LS, and indicates that the required entry specifies messages on a particular session for the specified LS. The structure contains the following three values, which are returned in the SESSION_STATS section of a QUERY_SESSION verb:

filter.lfsid.uu.s.sidh

Session ID high byte.

filter.lfsid.uu.s.sidl

Session ID low byte.

filter.lfsid.odai

Origin Destination Assignor Indicator.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer contains the following parameters:

overlay_size

The size of the returned dlc_trace_data structure, and therefore the offset to the start of the next entry in the data buffer.

dlc_trace_filter.resource_type

The type of resource being traced. This can take one of the following values:

ALL_RESOURCES

The entry specifies tracing options for all DLCs, ports, and LSs.

AP_DLC

The entry specifies tracing options for the DLC named in *resource_name*, and for all ports and LSs that use this DLC.

AP_PORT

The entry specifies tracing options for the port named in *resource_name*, and for all LSs that use this port.

AP_LS

The entry specifies tracing options for the LS named in *resource_name* (or for a particular LFSID on this LS).

AP_PORT_DEFINED_LS

The entry specifies tracing options for the port named in *resource_name*, and for all defined LSs (but not implicit LSs) that use this port.

AP_PORT_IMPLICIT_LS

The entry specifies tracing options for the port named in *resource_name*, and for all implicit LSs (but not

defined LSs) that use this port.

dlc_trace_filter.resource_name

The name of the DLC, port, or LS being traced.

dlc_trace_filter.lfsid

The Local Form Session Identifier for a session on the specified LS. This is only valid for *resource_type* AP_LS, and indicates that only messages on this session are to be traced. The structure contains the following three values, which are returned in the SESSION_STATS section of a QUERY_SESSION verb:

dlc_trace_filter.lfsid.uu.s.sidh

Session ID high byte.

dlc_trace_filter.lfsid.uu.s.sidl

Session ID low byte.

dlc_trace_filter.lfsid.odai

Origin Destination Assignor Indicator.

dlc_trace_filter.message_type

The type of messages being traced for the specified resource or session. This parameter is set to AP_TRACE_ALL to trace all messages, or to one or more of the following values (combined using a logical OR):

AP_TRACE_XID

XID messages

AP_TRACE_SC

Session Control RUs

AP_TRACE_DFC

Data Flow Control RUs

AP_TRACE_FMD

FMD messages

AP_TRACE_NLP

(this message type is currently not used)

AP_TRACE_NC

(this message type is currently not used)

AP_TRACE_SEGS

Non-BBIU segments that do not contain an RH

AP_TRACE_CTL

Messages other then MUs and XIDs

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns one of the following.

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LIST_TYPE

The *list_type* parameter specified a value that was not valid.

AP_INVALID_RESOURCE_TYPE

The *resource_type* parameter specified a value that was not valid.

AP_ALL_RESOURCES_NOT_DEFINED

The *resource_type* parameter was set to AP_ALL_RESOURCES, but there is no DLC_TRACE entry defined for tracing options on all resources.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLUR_DEFAULTS

The QUERY_DLUR_DEFAULTS verb allows the user to query the defaults defined using the DEFINE_DLUR_DEFAULTS verb.

VCB Structure

```
typedef struct query_dlur_defaults
{
    AP_UINT16         opcode;               /* verb operation code          */
    unsigned char     reserv2;              /* reserved                     */
    unsigned char     format;               /* reserved                     */
    AP_UINT16         primary_rc;           /* primary return code          */
    AP_UINT32         secondary_rc;         /* secondary return code        */
    unsigned char     description;          /* resource description         */
    unsigned char     dlus_name[17];        /* DLUS name                    */
    unsigned char     bkup_dlus_name[17];   /* Backup DLUS name             */
    unsigned char     reserv3;              /* reserved                     */
    AP_UINT16         dlus_retry_timeout    /* DLUS retry timeout           */
    AP_UINT16         dlus_retry_limit;     /* DLUS retry limit             */
    unsigned char     reserv4[16];          /* reserved                     */
 } QUERY_DLUR_DEFAULTS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLUR_DEFAULTS

description

Resource description. The length of this parameter is a multiple of four bytes and is nonzero.

dlus_name

Name of the DLUS node that is the default. This name is set to all zeros or a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8

A-string characters.

bkup_dlus_name

Name of the DLUS node that serves as the backup default. This name is set to all zeros or a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlus_retry_timeout

Interval in seconds between second and subsequent attempts to contact a DLUS. The interval between the initial attempt and the first retry is always one second.

dlus_retry_limit

Maximum number of retries after an initial failure to contact a DLUS. A value of 0xFFFF indicates that SNAplus2 retries indefinitely.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameter:

primary_rc AP_OK

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLUR_LU

QUERY_DLUR_LU returns information about active LUs that are using the DLUR feature of SNAplus2. This verb can be used to obtain information about a specific LU, or about multiple LUs, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_dlur_lu
{
    AP_UINT16         opcode;              /* verb operation code           */
    unsigned char     reserv2;             /* reserved                      */
    unsigned char     format;              /* reserved                      */
    AP_UINT16         primary_rc;          /* primary return code           */
    AP_UINT32         secondary_rc;        /* secondary return code         */
    unsigned char     *buf_ptr;            /* pointer to buffer             */
    AP_UINT32         buf_size;            /* buffer size                   */
    AP_UINT32         total_buf_size;      /* total buffer size required    */
    AP_UINT16         num_entries;         /* number of entries             */
    AP_UINT16         total_num_entries;   /* total number of entries       */
    unsigned char     list_options;        /* listing options               */
    unsigned char     reserv3;             /* reserved                      */
    unsigned char     lu_name[8];          /* LU name                       */
    unsigned char     pu_name[8];          /* PU name filter                */
    unsigned char     filter;              /* local / downstream filter     */
} QUERY_DLUR_LU;

typedef struct dlur_lu_summary
{
    AP_UINT16         overlay_size;        /* size of returned entry        */
    unsigned char     lu_name[8];          /* LU name                       */
} DLUR_LU_SUMMARY;

typedef struct dlur_lu_detail
{
    AP_UINT16         overlay_size;        /* size of returned entry        */
    unsigned char     lu_name[8];          /* LU name                       */
    unsigned char     pu_name[8];          /* PU name of owning PU          */
    unsigned char     dlus_name[17];       /* DLUS name if SSCP-LU session  */
```

```
                                        /*  active                      */
    unsigned char     lu_location;      /* downstream or local LU       */
    unsigned char     nau_address;      /* NAU address of LU            */
    unsigned char     plu_name[17];     /* PLU name if PLU-SLU session  */
                                        /*  active                      */
    unsigned char     reserv1[27];      /* reserved                     */
    unsigned char     rscv_len;         /* length of appended RSCV      */
} DLUR_LU_DETAIL;
```

NOTE The DLUR_LU_DETAIL structure may be followed by a Route Selection Control Vector (RSCV) as defined by SNA Formats. This control vector defines the session route through the network and is carried on the BIND. This RSCV is included only if the node's configuration (specified using DEFINE_NODE) indicates that RSCVs should be stored for DLUR sessions and if the PLU-SLU session is active.

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLUR_LU

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of DLUR LUs for which data should be returned. To request data for a specific LU rather

than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *pu_name* and *lu_name* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *pu_name* and *lu_name* parameters.

The list is ordered by *pu_name* and then by *lu_name*. For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

lu_name

Name of the LU for which information is required, or the name to be used as an index into the list of LUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces

if the name is shorter than 8 characters.

pu_name

PU name for which LU information is required. To list only information about LUs associated with a specific PU, specify the PU name. To obtain a complete list for all PUs, set this field to binary zeros. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

dlur_lu_summary.overlay_size

The size of the returned dlur_lu_summary structure, and therefore the offset to the start of the next entry in

the data buffer.

dlur_lu_summary.lu_name

Name of the LU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dlur_lu_detail.overlay_size

The size of the returned `dlur_lu_detail` structure, and therefore the offset to the start of the next entry in the data buffer.

dlur_lu_detail.lu_name

Name of the LU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dlur_lu_detail.pu_name

Name of PU associated with the LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

dlur_lu_detail.dlus_name

If the SSCP-LU session is active, this field contains the name of the DLUS node used by the LU; otherwise it is set to 17 binary zeros. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlur_lu_detail.lu_location

Location of LU.

This is set to `AP_INTERNAL`, indicating that the LU is on the local node.

dlur_lu_detail.nau_address

Network accessible unit address of the LU.

dlur_lu_detail.plu_name

If the PLU-SLU session is active, this field contains the name of the PLU; otherwise it is set to 17 binary zeros.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlur_lu_detail.rscv_len

Length of the RSCV that is appended to the dlur_lu_detail structure. If the node's configuration specifies that DLUR RSCVs are not stored, or if the PLU-SLU session is not active, this length is set to zero and no RSCV is included.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_FILTER_OPTION

The *filter* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLUR_PU

QUERY_DLUR_PU returns information about PUs that use the DLUR feature of SNAplus2.

This verb can be used to obtain information about a specific PU, or about multiple PUs, depending on the options used.

VCB Structure

```
typedef struct query_dlur_pu
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* primary return code          */
   AP_UINT32        secondary_rc;          /* secondary return code        */
   unsigned char    *buf_ptr;              /* pointer to buffer            */
   AP_UINT32        buf_size;              /* buffer size                  */
   AP_UINT32        total_buf_size;        /* total buffer size required   */
   AP_UINT16        num_entries;           /* number of entries            */
   AP_UINT16        total_num_entries;     /* total number of entries      */
   unsigned char    list_options;          /* listing options              */
   unsigned char    reserv3;               /* reserved                     */
   unsigned char    pu_name[8];            /* PU name                      */
   unsigned char    dlus_name[17];         /* fully-qualified DLUS name    */
   unsigned char    filter;                /* local / downstream filter    */
} QUERY_DLUR_PU;

typedef struct dlur_pu_summary
{
   AP_UINT16        overlay_size;          /* size of returned entry       */
   unsigned char    pu_name[8];            /* PU name                      */
   unsigned char    description[32];        /* resource description        */
   unsigned char    reserv1[16];           /* reserved                     */
} DLUR_PU_SUMMARY;

typedef struct dlur_pu_detail
{
   AP_UINT16        overlay_size;          /* size of returned entry       */
   unsigned char    pu_name[8];            /* PU name                      */
   unsigned char    description[32];        /* resource description        */
```

```
   unsigned char    initially_active;        /* is the PU initially active?  */
   unsigned char    reserv1[15];             /* reserved                     */
   unsigned char    defined_dlus_name[17];   /* defined DLUS name            */
   unsigned char    bkup_dlus_name[17];      /* backup DLUS name             */
   unsigned char    pu_id[4];                /* PU identifier                */
   unsigned char    pu_location;             /* downstream or local PU       */
   unsigned char    active_dlus_name[17];    /* active DLUS name             */
   unsigned char    ans_support;             /* auto network shutdown support*/
   unsigned char    pu_status;               /* status of the PU             */
   unsigned char    dlus_session_status;     /* status of the DLUS pipe      */
   unsigned char    reserv3;                 /* reserved                     */
   FQPCID           fqpcid;                  /* FQPCID used on pipe          */
   AP_UINT16        dlus_retry_timeout;      /* DLUR retry timeout           */
   AP_UINT16        dlus_retry_limit;        /* DLUR retry limit             */
} DLUR_PU_DETAIL

typedef struct fqpcid
{
   unsigned char    pcid[8];              /* procedure correlator identifier */
   unsigned char    fqcp_name[17];        /* originator's network qualified  */
                                          /*  CP name                        */
   unsigned char    reserve3[3];          /* reserved                        */
} FQPCID;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLUR_PU

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of DLUR PUs for which data should be returned. To request data for a specific PU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *pu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *pu_name* parameter.

The list is ordered by *pu_name*. For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

pu_name

Name of the PU for which information is required, or the name to be used as an index into the list of PUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces

if the name is shorter than 8 characters.

dlus_name

DLUS name for which PU information is required. To list only information about PUs associated with a specific DLUS, specify the DLUS name; a PU will be listed only if it has an SSCP-PU session to the specified DLUS node. To obtain a complete list for all DLUSs, set this field to binary zeros.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

filter

Specifies whether to filter the returned PUs according to their location.

For end node, this parameter is reserved (downstream DLUR PUs are not supported).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

dlur_pu_summary.overlay_size

The size of the returned dlur_pu_summary structure, and therefore the offset to the start of the next entry in the data buffer.

dlur_pu_summary.pu_name

Name of the PU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dlur_pu_summary.description

A null-terminated text string describing the PU, as specified in the definition of the PU.

dlur_pu_detail.overlay_size

The size of the returned dlur_pu_detail structure, and therefore the offset to the start of the next entry in the data buffer.

dlur_pu_detail.pu_name

Name of the PU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dlur_pu_detail.description

A null-terminated text string describing the PU, as specified in the definition of the PU.

dlur_pu_detail.initially_active

Specifies whether this PU is automatically started when the node is started. Possible values are:

AP_YES

The PU is automatically started when the node is

started.

AP_NO

The PU is not automatically started; it must be started manually.

dlur_pu_detail.defined_dlus_name

Name of DLUS node, defined by either a DEFINE_INTERNAL_PU verb or a DEFINE_LS verb (with *dspu_services* set to AP_DLUR).

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlur_pu_detail.bkup_dlus_name

Name of backup DLUS node, defined by either a DEFINE_INTERNAL_PU verb or a DEFINE_LS verb (with *dspu_services* set to AP_DLUR).

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlur_pu_detail.pu_id

PU identifier, either defined on DEFINE_INTERNAL_PU or obtained in an XID from a downstream PU. This is a 4-byte hexadecimal string, consisting of a block number (3 hexadecimal digits) and a node number (5 hexadecimal digits).

dlur_pu_detail.pu_location

Location of PU.

This is set to AP_INTERNAL, indicating that the PU is on the local node.

dlur_pu_detail.active_dlus_name

Name of DLUS node that the PU is currently using. If the SSCP-PU session is not active, this field will be set

to all binary zeros.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlur_pu_detail.ans_support

Auto Network Shutdown support, as sent to DLUR from the DLUS at SSCP-PU activation. It specifies whether link-level contact should be continued if the subarea node initiates an auto network shutdown procedure for the SSCP controlling the PU. Possible values are:

AP_CONT

Continue link-level contact

AP_STOP

Stop link-level contact.

This field is reserved if the SSCP-LU session is inactive.

dlur_pu_detail.pu_status

Status of the PU (as seen by DLUR). Possible values are:

AP_RESET

The PU is in reset state.

AP_PEND_ACTPU

The PU is waiting for an ACTPU from the host.

AP_PEND_ACTPU_RSP

Having forwarded an ACTPU to the PU, DLUR is now waiting for the PU to respond to it.

AP_ACTIVE

The PU is active.

AP_PEND_DACTPU_RSP

Having forwarded a DACTPU to the PU, DLUR is waiting for the PU to respond to it.

AP_PEND_INOP

DLUR is waiting for all necessary events to complete before it deactivates the PU.

dlur_pu_detail.dlus_session_status

Status of the DLUS pipe currently being used by the PU. Possible values are:

AP_PENDING_ACTIVE

The pipe is in the process of being activated.

AP_ACTIVE

The pipe is active.

AP_PENDING_INACTIVE

The pipe is in the process of being deactivated.

AP_INACTIVE

The pipe is not active.

dlur_pu_detail.fqpcid.pcid

Procedure Correlator ID used on the pipe. This is an 8-byte hexadecimal string. If the SSCP-PU session is not active this field will be set to binary zeros.

dlur_pu_detail.fqpcid.fqcp_name

Fully qualified Control Point name used on the pipe. If the SSCP-PU session is not active this field will be set to binary zeros.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

The combination of the *pcid* and *fqcp_name* parameters uniquely identify each PU whose sessions are being routed using DLUR. The *fqcp_name* parameter is the CP name of either the DLUR or DLUS

node, depending on which node initiated the SSCP-PU session activation.

dlur_pu_detail.dlus_retry_timeout

The interval in seconds between the second and subsequent attempts to contact the DLUS specified by the *def_data.dlus_name* and *def_data.bkup_dlus_name* parameters. The interval between the first and second attempts is always 1 second. If zero is specified, then the defaults specified using the DEFINE_DLUR_DEFAULTS verb are used. .

dlur_pu_detail.dlus_retry_limit

Number of attempts to recontact a DLUS after an initial failure. A value of zero indicates that the value from the DEFINE_DLUR_DEFAULTS verb is used. If 0xFFFF is returned, SNAplus2 will retry indefinitely.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *pu_name* parameter was not valid.

AP_INVALID_FILTER_OPTION

The *filter* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DLUS

QUERY_DLUS returns information about DLUS nodes known to the DLUR feature of SNAplus2. This verb returns pipe statistics (SSCP-PU and SSCP-LU session statistics); the QUERY_ISR_SESSION verb may be used to obtain PLU-SLU session statistics.

This verb can be used to obtain information about a specific DLUS, or about multiple DLUSs, depending on the options used.

If this verb is issued to an inactive node, it returns information only on DLUS nodes defined using DEFINE_INTERNAL_PU or DEFINE_DLUR_DEFAULTS; if it is issued to a running node, it also returns information about active DLUS nodes. It does not return information about the backup DLUS that was defined using DEFINE_DLUR_DEFAULTS, unless this DLUS is active.

VCB Structure

```
typedef struct query_dlus
{
   AP_UINT16       opcode;                /* verb operation code          */
   unsigned char   reserv2;               /* reserved                     */
   unsigned char   format;                /* reserved                     */
   AP_UINT16       primary_rc;            /* primary return code          */
   AP_UINT32       secondary_rc;          /* secondary return code        */
   unsigned char   *buf_ptr;              /* pointer to buffer            */
   AP_UINT32       buf_size;              /* buffer size                  */
   AP_UINT32       total_buf_size;        /* total buffer size required   */
   AP_UINT16       num_entries;           /* number of entries            */
   AP_UINT16       total_num_entries;     /* total number of entries      */
   unsigned char   list_options;          /* listing options              */
   unsigned char   reserv3;               /* reserved                     */
   unsigned char   dlus_name[17];         /* fully-qualified DLUS name    */
} QUERY_DLUS;

typedef struct dlus_data
{
   AP_UINT16       overlay_size;          /* size of returned entry       */
   unsigned char   dlus_name[17];         /* fully qualified DLUS name    */
   unsigned char   is_default;            /* is the DLUS the default      */
   unsigned char   is_backup_default;     /* is DLUS the backup default   */
   unsigned char   pipe_state;            /* state of CPSVRMGR pipe       */
```

```
   AP_UINT16       num_active_pus;          /* num of active PUs using pipe  */
   PIPE_STATS      pipe_stats;              /* pipe statistics               */
} DLUS_DATA;

typedef struct pipe_stats
{
   AP_UINT32       reqactpu_sent;           /* REQACTPUs sent to DLUS        */
   AP_UINT32       reqactpu_rsp_received;   /* RSP(REQACTPU)s received       */
                                            /*  from DLUS                    */
   AP_UINT32       actpu_received;          /* ACTPUs received from DLUS     */
   AP_UINT32       actpu_rsp_sent;          /* RSP(ACTPU)s sent to DLUS      */
   AP_UINT32       reqdactpu_sent;          /* REQDACTPUs sent to DLUS       */
   AP_UINT32       reqdactpu_rsp_received;  /* RSP(REQDACTPU)s received      */
                                            /*  from DLUS                    */
   AP_UINT32       dactpu_received;         /* DACTPUs received from DLUS    */
   AP_UINT32       dactpu_rsp_sent;         /* RSP(DACTPU)s sent to DLUS     */
   AP_UINT32       actlu_received;          /* ACTLUs received from DLUS     */
   AP_UINT32       actlu_rsp_sent;          /* RSP(ACTLU)s sent to DLUS      */
   AP_UINT32       dactlu_received;         /* DACTLUs received from DLUS    */
   AP_UINT32       dactlu_rsp_sent;         /* RSP(DACTLU)s sent to DLUS     */
   AP_UINT32       sscp_pu_mus_rcvd;        /* MUs for SSCP-PU sessions rcvd */
   AP_UINT32       sscp_pu_mus_sent;        /* MUs for SSCP-PU sessions sent */
   AP_UINT32       sscp_lu_mus_rcvd;        /* MUs for SSCP-LU sessions rcvd */
   AP_UINT32       sscp_lu_mus_sent;        /* MUs for SSCP-LU sessions sent */
} PIPE_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DLUS

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of DLUSs for which data should be returned. To request data for a specific DLUS rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *dlus_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *dlus_name* parameter.

The list is ordered by *dlus_name*. For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

dlus_name

Name of the DLUS for which information is required, or the name to be used as an index into the list of DLUSs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

dlus_data.overlay_size

The size of the returned dlus_data structure, and therefore the offset to the start of the next entry in the data buffer.

dlus_data.dlus_name

Name of DLUS. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

dlus_data.is_default

Specifies whether the DLUS node has been designated

as the default by a DEFINE_DLUR_DEFAULTS verb (AP_YES or AP_NO).

dlus_data.is_backup_default

Specifies whether the DLUS node has been designated as the backup default by a DEFINE_DLUR_DEFAULTS verb (AP_YES or AP_NO).

dlus_data.pipe_state

State of the pipe to the DLUS. Possible values are:

AP_PENDING_ACTIVE

The pipe is in the process of being activated.

AP_ACTIVE

The pipe is active.

AP_PENDING_INACTIVE

The pipe is in the process of being deactivated.

AP_INACTIVE

The pipe is not active.

dlus_data.num_active_pus

Number of PUs currently using the pipe to the DLUS.

dlus_data.pipe_stats.reqactpu_sent

Number of REQACTPUs sent to DLUS over the pipe.

dlus_data.pipe_stats.reqactpu_rsp_received

Number of RSP(REQACTPU)s received from DLUS over the pipe.

dlus_data.pipe_stats.actpu_received

Number of ACTPUs received from DLUS over the pipe.

dlus_data.pipe_stats.actpu_rsp_sent

Number of RSP(ACTPU)s sent to DLUS over the pipe.

dlus_data.pipe_stats.reqdactpu_sent

Number of REQDACTPUs sent to DLUS over the pipe.

dlus_data.pipe_stats.reqdactpu_rsp_received

Number of RSP(REQDACTPU)s received from DLUS over the pipe.

dlus_data.pipe_stats.dactpu_received

Number of DACTPUs received from DLUS over the pipe.

dlus_data.pipe_stats.dactpu_rsp_sent

Number of RSP(DACTPU)s sent to DLUS over the pipe.

dlus_data.pipe_stats.actlu_received

Number of ACTLUs received from DLUS over the pipe.

dlus_data.pipe_stats.actlu_rsp_sent

Number of RSP(ACTLU)s sent to DLUS over the pipe.

dlus_data.pipe_stats.dactlu_received

Number of DACTLUs received from DLUS over the pipe.

dlus_data.pipe_stats.dactlu_rsp_sent

Number of RSP(DACTLU)s sent to DLUS over the pipe.

dlus_data.pipe_stats.sscp_pu_mus_rcvd

Number of SSCP-PU MUs received from DLUS over the pipe.

dlus_data.pipe_stats.sscp_pu_mus_sent

Number of SSCP-PU MUs sent to DLUS over the pipe.

dlus_data.pipe_stats.sscp_lu_mus_rcvd

Number of SSCP-LU MUs received from DLUS over the pipe.

dlus_data.pipe_stats.sscp_lu_mus_sent

Number of SSCP-LU MUs sent to DLUS over the pipe.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DLUS_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *dlus_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support DLUR; this is defined by the *dlur_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DOMAIN_CONFIG_FILE

QUERY_DOMAIN_CONFIG_FILE returns the header information included in the SNAplus2 domain configuration file (the SNAplus2 version number, the revision level of the file, and an optional comment string supplied on DEFINE_DOMAIN_CONFIG_FILE).

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct query_domain_config_file
{
    AP_UINT16             opcode;              /* verb operation code       */
    unsigned char         reserv2;             /* reserved                  */
    unsigned char         format;              /* reserved                  */
    AP_UINT16             primary_rc;          /* primary return code       */
    AP_UINT32             secondary_rc;        /* secondary return code     */
    CONFIG_FILE_HEADER    hdr;
} QUERY_DOMAIN_CONFIG_FILE;


typedef struct config_file_header
{
    AP_UINT16             major_version;       /* major version number      */
    AP_UINT16             minor_version;       /* minor version number      */
    AP_UINT16             update_release;      /* update release            */
    AP_UINT32             revision_level;      /* file revision number      */
    unsigned char         comment[100];        /* optional comment string   */
    unsigned char         updating;            /* reserved                  */
} CONFIG_FILE_HEADER;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_DOMAIN_CONFIG_FILE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

hdr.major_version,
hdr.minor_version, hdr.update_release

The internal version identifier of the release of SNAplus2 that was used to create this file.

hdr.revision_level

The revision level of the file (stored internally by SNAplus2).

hdr.comment

An optional comment string containing information about the file. This is an ASCII string of 0-99 characters, followed by a null character.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DOWNSTREAM_LU

QUERY_DOWNSTREAM_LU returns information about downstream LUs that use PU concentration or DLUR or both. This information is structured as determined data (data gathered dynamically during execution, returned only if the node is active) and defined data (data supplied on DEFINE_DOWNSTREAM_LU). For DLUR-supported LUs, implicitly defined data is put in place when the downstream LU is activated.

This verb can be used to obtain information about a specific LU, or about multiple LUs, depending on the options used.

VCB Structure

```
typedef struct query_downstream_lu
{
   AP_UINT16        opcode;                 /* verb operation code         */
   unsigned char    reserv2;                /* reserved                    */
   unsigned char    format;                 /* reserved                    */
   AP_UINT16        primary_rc;             /* primary return code         */
   AP_UINT32        secondary_rc;           /* secondary return code       */
   unsigned char    *buf_ptr;               /* pointer to buffer           */
   AP_UINT32        buf_size;               /* buffer size                 */
   AP_UINT32        total_buf_size;         /* total buffer size required  */
   AP_UINT16        num_entries;            /* number of entries           */
   AP_UINT16        total_num_entries;      /* total number of entries     */
   unsigned char    list_options;           /* listing options             */
   unsigned char    reserv3;                /* reserved                    */
   unsigned char    dslu_name[8];           /* Downstream LU name          */
   unsigned char    dspu_name[8];           /* Downstream PU name filter   */
   unsigned char    dspu_services;          /* services provided to LU     */
} QUERY_DOWNSTREAM_LU;

typedef struct downstream_lu_summary
{
   AP_UINT16        overlay_size;           /* size of returned entry      */
   unsigned char    dslu_name[8];           /* LU name                     */
   unsigned char    dspu_name[8];           /* PU name                     */
   unsigned char    description[32];        /* resource description        */
   unsigned char    reserv1[16];            /* reserved                    */
   unsigned char    dspu_services;          /* Type of services provided   */
                                            /*  to downstream LU           */
```

```
   unsigned char    nau_address;              /* NAU address                 */
   unsigned char    lu_sscp_sess_active;      /* Is LU-SSCP session active   */
   unsigned char    plu_sess_active;          /* Is PLU-SLU session active   */
} DOWNSTREAM_LU_SUMMARY;

typedef struct downstream_lu_detail
{
   AP_UINT16        overlay_size;             /* size of returned entry      */
   unsigned char    dslu_name[8];             /* LU name                     */
   unsigned char    reserv1[2];               /* reserved                    */
   DOWNSTREAM_LU_DET_DATA  det_data;          /* Determined data             */
   DOWNSTREAM_LU_DEF_DATA  def_data;          /* Defined data                */
} DOWNSTREAM_LU_DETAIL;

typedef struct downstream_lu_det_data
{
   unsigned char    lu_sscp_sess_active;      /* Is LU-SSCP session active   */
   unsigned char    plu_sess_active;          /* Is PLU-SLU session active   */
   unsigned char    dspu_services;            /* Type of service provided to */
                                              /*  downstream node            */
   unsigned char    reserv1;                  /* reserved                    */
   SESSION_STATS    lu_sscp_stats;            /* LU-SSCP session statistics  */
   SESSION_STATS    ds_plu_stats;             /* Downstream PLU-SLU session  */
                                              /*  statistics                 */
   SESSION_STATS    us_plu_stats;             /* Upstream PLU-SLU session    */
                                              /*  statistics                 */
   unsigned char    host_lu_name[8];          /* Determined host LU name     */
   unsigned char    host_pu_name[8];          /* Determined host PU name     */

   unsigned char    reserva[4];               /* reserved                    */
} DOWNSTREAM_LU_DET_DATA;

typedef struct downstream_lu_def_data
{
   unsigned char    description[32];          /* resource description        */
   unsigned char    reserv1[16];              /* reserved                    */
   unsigned char    nau_address;              /* downstream LU nau address   */
   unsigned char    dspu_name[8];             /* Downstream PU name          */
   unsigned char    host_lu_name[8];          /* Host LU or Pool name        */
   unsigned char    allow_timeout;            /* Allow timeout of host LU    */
   unsigned char    delayed_logon;            /* Allow delayed logon to      */
                                              /*  host LU                    */
 DOWNSTREAM_LU_DEF_DATA;

typedef struct session_stats
{
  AP_UINT16  rcv_ru_size;              /* session receive RU size          */
  AP_UINT16  send_ru_size;             /* session send RU size             */
```

```
  AP_UINT16  max_send_btu_size;         /* maximum send BTU size             */
  AP_UINT16  max_rcv_btu_size;          /* maximum rcv BTU size              */
  AP_UINT16  max_send_pac_win;          /* maximum send pacing window size   */
  AP_UINT16  cur_send_pac_win;          /* current send pacing window size   */
  AP_UINT16  max_rcv_pac_win;           /* maximum receive pacing window size*/
  AP_UINT16  cur_rcv_pac_win;           /* current receive pacing window size*/
  AP_UINT32   send_data_frames;         /* number of data frames sent        */
  AP_UINT32   send_fmd_data_frames;     /* num fmd data frames sent          */
  AP_UINT32   send_data_bytes;          /* number of data bytes sent         */
  AP_UINT32   rcv_data_frames;          /* number of data frames received    */
  AP_UINT32   rcv_fmd_data_frames;      /* num fmd data frames received      */
  AP_UINT32   rcv_data_bytes;           /* number of data bytes received     */
  unsigned char   sidh;                 /* session ID high byte (from LFSID) */
  unsigned char   sidl;                 /* session ID low byte (from LFSID)  */
  unsigned char   odai;                 /* ODAI bit set                      */
  unsigned char   ls_name[8];           /* Link station name                 */
  unsigned char   pacing_type;          /* type of pacing in use             */
} SESSION_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_DOWNSTREAM_LU

buf_ptr A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size Size of the supplied data buffer.

num_entries Maximum number of downstream LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *dspu_name* and *dslu_name* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *dspu_name* and *dslu_name* parameters.

The list is ordered by *dspu_name* and then by *dslu_name*. For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs".

dslu_name Name of the LU for which information is required, or the name to be used as an index into the list of LUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dspu_name PU name for which LU information is required (as specified on DEFINE_LS). To list only information about LUs associated with a specific PU, specify the PU name. To obtain a complete list for all PUs, set this field to binary zeros. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dspu_services DSPU services filter. When the verb is issued to a running node, this parameter specifies whether to filter the returned information by the type of services provided to the LUs. Possible values are:

AP_PU_CONCENTRATION

Return information only on downstream LUs served by PU concentration.

AP_DLUR

Return information only on downstream LUs served by DLUR.

AP_NONE

Return information about all downstream LUs.

When the node is not running, this parameter is ignored; SNAplus2 returns information about all downstream LUs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

downstream_lu_summary.overlay_size

The size of the returned downstream_lu_summary structure, and therefore the offset to the start of the

next entry in the data buffer.

downstream_lu_summary.dslu_name

Name of the LU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

downstream_lu_summary.dspu_name

Name of the PU associated with the LU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

downstream_lu_summary.description

A null-terminated text string describing the downstream LU, as specified in the definition of the downstream LU.

For a DLUR-supported LU, this parameter is reserved.

downstream_lu_summary.dspu_services

When the verb is issued to a running node, this parameter specifies the services provided by the local node to the downstream LU.

Possible values are:

AP_PU_CONCENTRATION

Downstream LU is served by PU concentration.

AP_DLUR

Downstream LU is served by DLUR.

downstream_lu_summary.nau_address

Network accessible unit address of the LU.

downstream_lu_summary.lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES	The session is active.
AP_NO	The session is not active.

downstream_lu_summary.plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES	The session is active.
AP_NO	The session is not active.

downstream_lu_detail.overlay_size

The size of the returned downstream_lu_detail structure, and therefore the offset to the start of the next entry in the data buffer.

downstream_lu_detail.dslu_name

Name of the LU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

downstream_lu_detail.det_data.lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES	The session is active.
AP_NO	The session is not active.

downstream_lu_detail.det_data.plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES	The session is active.
AP_NO	The session is not active.

downstream_lu_detail.det_data.dspu_services

When the verb is issued to a running node, this parameter specifies the services provided by the local node to the downstream LU.

Possible values are:

AP_PU_CONCENTRATION

Downstream LU is served by PU concentration.

AP_DLUR

Downstream LU is served by DLUR.

A `session_stats` structure is included for each of the three sessions (LU-SSCP session, downstream PLU-SLU session, and upstream PLU-SLU session). The fields in this structure are as follows:

`rcv_ru_size`

Maximum receive RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

`send_ru_size`

Maximum send RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

`max_send_btu_size`

Maximum BTU size that can be sent.

`max_rcv_btu_size`

Maximum BTU size that can be received.

`max_send_pac_win`

Maximum size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

`cur_send_pac_win`

Current size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

`max_rcv_pac_win`

Maximum size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

`cur_rcv_pac_win`

Current size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

`send_data_frames`

Number of normal flow data frames sent.

`send_fmd_data_frames`

Number of normal flow FMD data frames sent.

send_data_bytes

Number of normal flow data bytes sent.

rcv_data_frames

Number of normal flow data frames received.

rcv_fmd_data_frames

Number of normal flow FMD data frames received.

rcv_data_bytes

Number of normal flow data bytes received.

sidh

Session ID high byte. (In the upstream PLU-SLU session statistics for an LU served by PU concentration, this parameter is reserved.)

sidl

Session ID low byte. (In the upstream PLU-SLU session statistics for an LU served by PU concentration, this parameter is reserved.)

odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station. (In the upstream PLU-SLU session statistics for an LU served by PU concentration, this parameter is reserved.)

ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters. (In the upstream PLU-SLU session statistics for an LU served by PU concentration, this parameter is reserved.)

pacing_type

The type of receive pacing in use on this session.

Possible values are:

```
AP_NONE
AP_PACING_FIXED
```

downstream_lu_detail.det_data.host_lu_name

Name of the host LU to which the downstream LU is mapped, or to which it was mapped when the PLU-SLU session was last active. This parameter value may differ from *def_data.host_lu_name* because *def_data.host_lu_name* can be the name of a host LU pool.

downstream_lu_detail.det_data.host_pu_name

Name of the host PU to which the downstream LU is mapped, or to which it was mapped when the PLU-SLU session was last active.

downstream_lu_detail.def_data.description

A null-terminated text string describing the downstream LU, as specified in the definition of the downstream LU. For a DLUR-supported LU, this parameter is reserved.

downstream_lu_detail.def_data.nau_address

Network accessible unit address of the downstream LU.

downstream_lu_detail.def_data.dspu_name

Name of the downstream PU associated with this LU (as specified on the DEFINE_LS verb). This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

downstream_lu_detail.def_data.host_lu_name

Name of the host LU or host LU pool that the downstream LU uses. This is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

This field is reserved for DLUR-served downstream LUs.

downstream_lu_detail.allow_timeout

Specifies whether this downstream LU allows its session with the upstream LU to timeout. Possible values are:

AP_YES

This downstream LU allows its session with the upstream LU to timeout.

AP_NO

This downstream LU does not allow its session with the upstream LU to timeout.

downstream_lu_detail.delayed_logon

Specifies whether this downstream LU uses delayed logon (the upstream LU is not activated until the user requests that it be activated). Possible values are:

AP_YES

This downstream LU uses delayed logon.

AP_NO

This downstream LU does not use delayed logon.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_PU_TYPE

The PU specified by the *dspu_name* parameter is not a downstream PU.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration or DLUR; this is defined by the *pu_conc_supported* and *dlur_supported* parameters on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DOWNSTREAM_PU

QUERY_DOWNSTREAM_PU returns information about downstream PUs that use PU concentration or DLUR or both. This verb can be used to obtain information about a specific PU or about multiple PUs, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_downstream_pu
{
  AP_UINT16        opcode;              /* verb operation code             */
  unsigned char    reserv2;             /* reserved                        */
  unsigned char    format;              /* reserved                        */
  AP_UINT16        primary_rc;          /* primary return code             */
  AP_UINT32        secondary_rc;        /* secondary return code           */
  unsigned char    *buf_ptr;            /* pointer to buffer               */
  AP_UINT32        buf_size;            /* buffer size                     */
  AP_UINT32        total_buf_size;      /* total buffer size required      */
  AP_UINT16        num_entries;         /* number of entries               */
  AP_UINT16        total_num_entries;   /* total number of entries         */
  unsigned char    list_options;        /* listing options                 */
  unsigned char    reserv3;             /* reserved                        */
  unsigned char    dspu_name[8];        /* Downstream PU name filter       */
  unsigned char    dspu_services;       /* services provided to PU         */
} QUERY_DOWNSTREAM_PU;

typedef struct downstream_pu_data
{
  AP_UINT16        overlay_size;        /* size of returned entry          */
  unsigned char    dspu_name[8];        /* PU name                         */
  unsigned char    description[32];     /* resource description            */
  unsigned char    reserv1[16];         /* reserved                        */
  unsigned char    ls_name[8];          /* Link name                       */
  unsigned char    pu_sscp_sess_active; /* Is the PU-SSCP session active   */
  unsigned char    dspu_services;       /* DSPU service type               */
  unsigned char    reserv1[2];          /* reserved                        */
  SESSION_STATS    pu_sscp_stats;       /* SSCP-PU session statistics      */
  unsigned char    reserva[20];         /* reserved                        */
} DOWNSTREAM_PU_DATA;

typedef struct session_stats
```

```
{
  AP_UINT16   rcv_ru_size;              /* session receive RU size          */
  AP_UINT16   send_ru_size;             /* session send RU size             */
  AP_UINT16   max_send_btu_size;        /* maximum send BTU size            */
  AP_UINT16   max_rcv_btu_size;         /* maximum rcv BTU size             */
  AP_UINT16   max_send_pac_win;         /* maximum send pacing window size  */
  AP_UINT16   cur_send_pac_win;         /* current send pacing window size  */
  AP_UINT16   max_rcv_pac_win;          /* maximum receive pacing window    */
                                        /*  size                            */
  AP_UINT16   cur_rcv_pac_win;          /* current receive pacing window    */
                                        /*  size                            */
  AP_UINT32    send_data_frames;        /* number of data frames sent       */
  AP_UINT32    send_fmd_data_frames;    /* num fmd data frames sent         */
  AP_UINT32    send_data_bytes;         /* number of data bytes sent        */
  AP_UINT32    rcv_data_frames;         /* number of data frames received   */
  AP_UINT32    rcv_fmd_data_frames;     /* num fmd data frames received     */
  AP_UINT32    rcv_data_bytes;          /* number of data bytes received    */
  unsigned char    sidh;                /* session ID high byte (from LFSID)*/
  unsigned char    sidl;                /* session ID low byte (from LFSID) */
  unsigned char    odai;                /* ODAI bit set                     */
  unsigned char    ls_name[8];          /* Link station name                */
  unsigned char    pacing_type;         /* type of pacing in use            */
} SESSION_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_DOWNSTREAM_PU

buf_ptr A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size Size of the supplied data buffer.

num_entries Maximum number of downstream PUs for which data should be returned. To request data for a specific PU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *dspu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *dspu_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see List Options For QUERY_* Verbs.

dspu_name Name of the PU for which information is required (as specified on DEFINE_LS), or the name to be used as an index into the list of PUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

dspu_services DSPU services filter. Specifies whether to filter the returned information by the type of services provided to the PUs. Possible values are:

AP_PU_CONCENTRATION

Return information only on downstream PUs served by PU concentration.

AP_DLUR

Return information only on downstream PUs served by DLUR.

AP_NONE

Return information about all downstream PUs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

downstream_pu_data.overlay_size

The size of the returned downstream_pu_data structure, and therefore the offset to the start of the next entry in the data buffer.

downstream_pu_data.dspu_name

Name of the downstream PU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

downstream_pu_data.description

A null-terminated text string describing the LS to the downstream PU, as specified in the definition of the LS.

downstream_pu_data.ls_name

Name of the LS used to access the downstream PU. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

downstream_pu_data.pu_sscp_sess_active

Specifies whether the PU-SSCP session to the downstream PU is active. Possible values are:

AP_YES — The session is active.

AP_NO — The session is not active.

downstream_pu_data.dspu_services

Specifies the type of services provided to the PU.

Possible values are:

AP_PU_CONCENTRATION

Downstream PU is served by PU concentration.

AP_DLUR

Downstream PU is served by DLUR.

downstream_pu_data.pu_sscp_stats.rcv_ru_size

Maximum receive RU size; this field is reserved (and set to zero) if the downstream PU is served by PU concentration.

downstream_pu_data.pu_sscp_stats.send_ru_size

Maximum send RU size; this field is reserved (and set to zero) if the downstream PU is served by PU concentration.

downstream_pu_data.pu_sscp_stats.max_send_btu_size

Maximum BTU size that can be sent.

downstream_pu_data.pu_sscp_stats.max_rcv_btu_size

Maximum BTU size that can be received.

downstream_pu_data.pu_sscp_stats.max_send_pac_win

Reserved (always set to zero).

downstream_pu_data.pu_sscp_stats.cur_send_pac_win

Reserved (always set to zero).

downstream_pu_data.pu_sscp_stats.max_rcv_pac_win

Reserved (always set to zero).

downstream_pu_data.pu_sscp_stats.cur_rcv_pac_win

Reserved (always set to zero).

downstream_pu_data.pu_sscp_stats.send_data_frames

Number of normal flow data frames sent.

downstream_pu_data.pu_sscp_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

downstream_pu_data.pu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

downstream_pu_data.pu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

downstream_pu_data.pu_sscp_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

downstream_pu_data.pu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

downstream_pu_data.pu_sscp_stats.sidh

Session ID high byte.

downstream_pu_data.pu_sscp_stats.sidl

Session ID low byte.

downstream_pu_data.pu_sscp_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station.

downstream_pu_data.pu_sscp_stats.ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters.

downstream_pu_data.pacing_type

The type of receive pacing in use on the PU-SSCP. This parameter will always be set to AP_NONE.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *dspu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support PU concentration or DLUR; this is defined by the *pu_conc_supported* and *dlur_supported* parameterson the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_DSPU_TEMPLATE

The QUERY_DSPU_TEMPLATE verb returns information about defined downstream PU templates used for PU concentration over implicit links.

This verb can be used to obtain information about a specific downstream PU template, or about a number of downstream PU templates, depending on the options used. To obtain information about a specific downstream PU template or multiple downstream PU templates, set the *template_name* parameter. The *template_name* parameter is ignored if the *list_options* parameter is set to AP_FIRST_IN_LIST.

VCB Structure

```
typedef struct query_dspu_template
{
   AP_UINT16  opcode;                /* verb operation code                */
   unsigned char   reserv2;          /* reserved                           */
   unsigned char   format;           /* reserved                           */
   AP_UINT16  primary_rc;            /* primary return code                */
   AP_UINT32   secondary_rc;         /* secondary return code              */
   unsigned char   *buf_ptr;         /* pointer to buffer                  */
   AP_UINT32   buf_size;             /* buffer size                        */
   AP_UINT32   total_buf_size;       /* total buffer size required         */
   AP_UINT16  num_entries;           /* number of entries                  */
   AP_UINT16  total_num_entries;     /* total number of entries            */
   unsigned char   list_options;     /* listing options                    */
   unsigned char   reserv3;          /* reserved                           */
   unsigned char   template_name[8]; /* name of DSPU template              */
 } QUERY_DSPU_TEMPLATE;

typedef struct dspu_template_data
{
   AP_UINT16       overlay_size;     /* size of returned entry             */
   unsigned char   template_name[8]; /* name of DSPU template              */
   unsigned char   description;      /* resource description               */
   unsigned char   reserv1[12];      /* reserved                           */
   AP_UINT16       max_instance;     /* max active template instance       */
   AP_UINT16       active_instance;  /* current active instances           */
   unsigned char   num_of_dslu_templates;  /* number of DSLU templates     */
 } DSPU_TEMPLATE_DATA;
```

Each dspu_template_data structure is followed by one or more

downstream LU templates; the number of the downstream LU templates is specified by the *number_of_dslu_templates* parameter. Each downstream LU template has the following format:

```
typedef struct dslu_template_data
{
   AP_UINT16  overlay_size;          /* size of this entry                */
   unsigned char   reserv1[2];       /* reserved                          */
   DSLU_TEMPLATE   dslu_template;    /* downstream LU template            */
} DSLU_TEMPLATE_DATA;

typedef struct dslu_template
{
   unsigned char   min_nau;          /* minimum NAU address in range      */
   unsigned char   max_nau;          /* maximum NAU address in range      */
   unsigned char   allow_timeout;    /* allow timeout of host LU?         */
   unsigned char   delayed_logon;    /* allow delayed logon to host LU    */
   unsigned char   reserv1[8];       /* reserved                          */
   unsigned char   host_lu[8];       /* host LU or pool name              */
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_DSPU_TEMPLATE

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of template for which data should be returned. To request data for a specific template rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *template_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *template_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

template_name

Name of the DSPU template for which information is required, or the name to be used as an index into the list. This is an 8-byte string in a locally displayable character set. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

num_entries

The number of entries actually returned. The last

entry may be incomplete; this is indicated by the *last_user_incomplete* parameter.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

dspu_template_data.overlay_size

The number of bytes in this entry, including any downstream LU templates, and the offset to the next entry returned (if any).

dspu_template_data.template_name

Name of the DSPU template.

dspu_template_data.description

Resource description, as defined on the DEFINE_DSPU_TEMPLATE verb.

dspu_template_data.max_instance

The maximum number of instances of the template which can be active simultaneously.

dspu_template_data.active_instance

The number of instances of the template which are currently active.

dspu_template_data.num_of_dslu_templates

Number of downstream LU templates for this downstream PU template. Following this parameter are *num_of_dslu_templates* entries, one for each DSLU template.

dslu_template_data.overlay_size

The number of bytes in this entry, and the offset to the next entry returned (if any).

dslu_template_data.min_nau

Minimum NAU address in the range of DSLU templates.

dslu_template_data.max_nau

Maximum NAU address in the range of DSLU templates.

dslu_template_data.allow_timeout

Indicates whether SNAplus2 is allowed to timeout host LUs used by this downstream LU if the session is left inactive for the timeout period specified on the host LU definition. Possible values are:

AP_YES

SNAplus2 is allowed to timeout host LUs used by this downstream LU.

AP_NO

SNAplus2 is not allowed to timeout host LUs used by this downstream LU.

dslu_template_data.delayed_logon

Indicates whether SNAplus2 delays connecting the downstream LU to the host LU until the first data is received from the downstream LU. Instead, a simulated logon screen is sent to the downstream LU. Possible values are:

AP_YES

SNAplus2 delays connecting the downstream LU to the host LU.

AP_NO

SNAplus2 does not delay connecting the downstream LU to the host LU.

dslu_template_data.host_lu_name

Name of the host LU or host LU pool onto which all the downstream LUs within the range will be mapped.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_TEMPLATE_NAME

The template specified in the *template_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_EMULATOR_USER_DEF

QUERY_EMULATOR_USER_DEF returns information about 3270 users or 5250 users defined in a SNAplus2 configuration file. It can return either summary or detailed information, about a single user or multiple users, depending on the options used. This verb returns information about the definition of this user in the configuration file, not about the user's current usage of the emulation program. You can use QUERY_3270_USER and QUERY_3270_USER_SESSIONS to obtain information about a user's current 3270 emulation usage.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct query_emulator_user_def
{
   AP_UINT16       opcode;                /* verb operation code            */
   unsigned char   reserv2;               /* reserved                       */
   unsigned char   format;                /* reserved                       */
   AP_UINT16       primary_rc;            /* primary return code            */
   AP_UINT32       secondary_rc;          /* secondary return code          */
   unsigned char   *buf_ptr;              /* pointer to buffer              */
   AP_UINT32       buf_size;              /* buffer size                    */
   AP_UINT32       total_buf_size;        /* total buffer size required     */
   AP_UINT16       num_entries;           /* number of entries              */
   AP_UINT16       total_num_entries;     /* total number of entries        */
   unsigned char   list_options;          /* listing options                */
   unsigned char   reserv3;               /* reserved                       */
   unsigned char   user_name[32];         /* 3270 user name                 */
   unsigned char   session_name[8];       /* session name                   */
   unsigned char   emulator_type;         /* type of entries to return      */
   unsigned char   reserv4;               /* reserved                       */
   AP_UINT32       num_init_sessions;     /* number of sessions for first   */
                                          /*  user when starting in middle  */
   AP_UINT32       num_last_sessions;     /* number of sessions on last     */
                                          /* detail overlay if last user    */
                                          /*  is incomplete                 */
   unsigned char   last_user_incomplete;  /* set to AP_YES if session       */
                                          /*  data for last user incomplete */
} QUERY_EMULATOR_USER_DEF;
```

```
typedef struct emulator_user_summary
{
   unsigned char          user_name[32];         /* user name                */
} EMULATOR_USER_SUMMARY;


typedef struct emulator_user_detail
{
   unsigned char          user_name[32];         /* user name                */
   AP_UINT32              num_filtered_sessions; /* total number of sessions */
                                                 /*  of the requested type   */
   EMULATOR_USER_DEF_DATA def_data;              /* user definition          */
} EMULATOR_USER_DETAIL;


typedef struct emulator_user_def_data
{
   unsigned char   description[32];    /* Description - null terminated      */
   unsigned char   reserv3[16];        /* reserved                           */
   unsigned char   style_file[9];      /* Initial style file name            */
                                       /*  - null terminated                 */
   unsigned char   reserv1[3];         /* reserved                           */
   AP_UINT32       num_sessions;       /* Number of sessions                 */
   AP_UINT32       max_act_sessions;   /* Maximum number of active sessions  */
   unsigned char   view_rtm;           /* Can user view RTM info?            */
   unsigned char   alert_permission;   /* Has user got ALERT permission?     */
   unsigned char   change_lu;          /* Can user change LU/Pool accessed?  */
   unsigned char   change_style;       /* Can user modify initial style file?*/
   unsigned char   user_is_group;      /* Does user_name specify a           */
                                       /*  HP-UX  group?                     */
   unsigned char   status;             /* reserved                           */
   unsigned char   reserv2;            /* reserved                           */
} EMULATOR_USER_DEF_DATA;


typedef struct session_def_data
{
   AP_UINT16       sub_overlay_size;   /* size of session_def_data structure */
   unsigned char   session_name[8];    /* Long session name                  */
   unsigned char   emulator_type;      /* Emulator type - 3270 or 5250       */
   unsigned char   reserv1;            /* reserved                           */
   unsigned char   description[32];    /* Session description                */
   unsigned char   reserv2[16];        /* reserved                           */
   union
   {
       SESSION_3270_DEF_DATA   def_data_3270;  /* definition of 3270 session */
       SESSION_5250_DEF_DATA   def_data_5250;  /* definition of 5250 session */
```

```
   } session_variant;
   AP_UINT32       reserv4;           /* reserved                          */
} SESSION_DEF_DATA;

typedef struct session_3270_def_data
{
   unsigned char   lu_name[8];        /* LU/pool name accessed             */
   unsigned char   session_type;      /* Session type - Model 2-5 or printer*/
   unsigned char   model_override;    /* Can the user override the model?  */
} SESSION_3270_DEF_DATA;

typedef struct session_5250_def_data
{
   unsigned char   local_lu_alias[8]; /* Local LU alias                    */
   unsigned char   plu_alias[8];      /* Partner LU alias                  */
   unsigned char   fqplu_name[17];    /* Fully-qualified partner LU name   */
   unsigned char   mode_name[8];      /* Mode name                         */
   unsigned char   session_type;      /* Session type - display or printer */
} SESSION_5250_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_EMULATOR_USER_DEF

sub_overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of users for which data should be

returned. If detailed information about user sessions is being returned, this number includes partial entries (for which a session name is specified, so that the returned data does not include the user definition or the user's first session).

To request data for a specific user rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

`list_options`

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

`AP_SUMMARY`

Summary information only.

`AP_DETAIL`

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

`AP_FIRST_IN_LIST`

Start at the first session for the first user in the list.

`AP_LIST_INCLUSIVE`

Start at the entry specified by the supplied user name and session name, or start at the first session for the specified user if no session name is specified.

`AP_LIST_FROM_NEXT`

If a session name is specified, start at the session immediately following the specified session. If no session name is specified, start at the first session for the specified user.

The list is ordered by user name, and then by session name (irrespective of session type) for each user. For more information about how the list is ordered and how

the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

user_name

The name of the user for whom information is required, or the name to be used as an index into the list of users. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an ASCII string of 1-32 characters, padded on the right with spaces if the name is shorter than 32 characters.

session_name

To return information starting with a specific session name for the specified user, set this parameter to the session name. To return information starting at the first session for the specified user, set this parameter to 8 binary zeros.

emulator_type

Specify whether to filter the returned information by session type. Possible values are:

AP_3270_SESSION

Return only information about 3270 sessions.

AP_5250_SESSION

Return only information about 5250 sessions.

AP_ALL

Return information about all sessions regardless of session type.

AP_NONE

Do not return any session_def_data structures. This value can be used with *list_options* set to AP_DETAIL, to obtain detailed information about each user's style file and 3270 permissions but to suppress detailed information about individual sessions.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned. The last entry may be incomplete; this is indicated by the *last_user_incomplete* parameter.

num_init_sessions

If the *session_name* parameter was set to a nonzero value, so that the information for the first user in the list does not start with the user's first session, this parameter indicates the number of session structures for this user that are included in the returned data. Otherwise, this parameter is not used.

num_last_sessions

If the *last_user_incomplete* parameter indicates that the data for the last user is incomplete, this parameter indicates the number of session structures for this user that are included in the returned data. (The *num_filtered_sessions* parameter returned for this user indicates the total number of session structures of the requested type that are available.) Otherwise, this parameter is not used.

last_user_incomplete

Specifies whether the information for the last user is

incomplete. Possible values are:

AP_YES

The complete data for the last user was too large to fit in the data buffer. At least one session structure is included, but there are further session structures that are not included in the data buffer. The *num_last_sessions* parameter indicates how many session structures have been returned; the application can issue further verbs to obtain the remaining data.

AP_NO

The data for the last user is complete.

Each entry in the data buffer consists of the following:

emulator_user_summary.user_name

The name of the user or group. This is an ASCII string of 1-32 characters.

emulator_user_detail.user_name

The name of the user or group. This is an ASCII string of 1-32 characters.

emulator_user_detail.num_filtered_sessions

The total number of sessions for this user of the type(s) specified by *emulator_type*. If *emulator_type* was set to AP_NONE, this parameter is zero.

emulator_user_detail.def_data

The details of the user, as defined in the configuration. This is followed by a number of session structures, defining the user's sessions and the user's remap list. The format of this information is the same as for the DEFINE_EMULATOR_USER verb, except for the following:

- The *sub_overlay_size*def_data parameter in the

  structure specifies the size of each returned session_def_data structure, and therefore the offset to the start of the next entry in the data buffer.

- The *num_sessions* parameter in the def_data structure defines the total number of sessions (3270, 5250, or both) defined for the user.
- If the *session_name* parameter was set to a nonzero value, the data for the first user will contain only the remaining session structures (starting from the requested entry), without the def_data structure.
- If the *last_user_incomplete* parameter is set to AP_YES, the total number of session structures returned for the last user will be as specified by the *num_last_sessions* parameter; this will be less than *num_sessions*.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_EMULATOR_USER

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *user_name* parameter did not match the name of any defined user.

AP_INVALID_SESSION_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *session_name* parameter did not match a session name defined for the specified user.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_FOCAL_POINT

QUERY_FOCAL_POINT returns information about the focal point for a specific Management Services category, or about multiple focal points, depending on the options used.

VCB Structure

```
typedef struct query_focal_point
{
   AP_UINT16       opcode;                /* verb operation code            */
   unsigned char   reserv2;               /* reserved                       */
   unsigned char   format;                /* reserved                       */
   AP_UINT16       primary_rc;            /* primary return code            */
   AP_UINT32       secondary_rc;          /* secondary return code          */
   unsigned char   *buf_ptr;              /* pointer to buffer              */
   AP_UINT32       buf_size;              /* buffer size                    */
   AP_UINT32       total_buf_size;        /* total buffer size required     */
   AP_UINT16       num_entries;           /* number of entries              */
   AP_UINT16       total_num_entries;     /* total number of entries        */
   unsigned char   list_options;          /* listing options                */
   unsigned char   reserv3;               /* reserved                       */
   unsigned char   ms_category[8];        /* name of MS category            */
} QUERY_FOCAL_POINT;


typedef struct fp_data
{
   AP_UINT16       overlay_size;          /* size of returned entry         */
   unsigned char   ms_appl_name[8];       /* focal point application name   */
   unsigned char   ms_category[8];        /* focal point category           */
   unsigned char   description[32];       /* resource description           */
   unsigned char   reserv1[16];           /* reserved                       */
   unsigned char   fp_fqcp_name[17];      /* focal point fully qualified    */
                                          /*  cp name                       */
   unsigned char   bkup_appl_name[8];     /* backup focal point             */
                                          /*  application name              */
   unsigned char   bkup_fp_fqcp_name[17]; /* backup fp fully qualified cp   */
                                          /*  name                          */
   unsigned char   implicit_appl_name[8]; /* implicit focal point appl name */
   unsigned char   implicit_fp_fqcp_name[17]; /* implicit fp fully qualified */
                                             /*  cp name                    */
   unsigned char   fp_type;               /* focal point type               */
   unsigned char   fp_status;             /* focal point status             */
```

```
   unsigned char   fp_routing;             /* type of MDS routing to use     */
   unsigned char   reserv1;                /* reserved                       */
   unsigned char   reserva[20];            /* reserved                       */
   AP_UINT16       number_of_appls;        /* number of applications         */
}FP_DATA;
```

Each `fp_data` structure is followed by one or more application names; the number of these is specified by the *`number_of_appls`* parameter. Each application name has the following format:

```
   unsigned char   appl_name[8];           /* application name               */
```

Supplied Parameters

The application supplies the following parameters:

`opcode`

`AP_QUERY_FOCAL_POINT`

`overlay_size`

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

`buf_ptr`

A pointer to a data buffer that SNAplus2 will use to return the requested information.

`buf_size`

Size of the supplied data buffer.

`num_entries`

Maximum number of entries for which data should be returned. To request data for a specific entry rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

`list_options`

The position in the list of focal points from which

SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *ms_category* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *ms_category* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

ms_category

Management Services category. This parameter is not used if *list_options* is set to AP_FIRST_IN_LIST.

This may be either one of the category names specified in the MS Discipline-Specific Application Programs table of *Systems Network Architecture: Management Services Reference* (see "Related Publications"), padded with EBCDIC spaces (0x40), or a user-defined category. A user-defined category name is an 8-byte type-1134 EBCDIC string, padded with EBCDIC spaces (0x40) if necessary.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

fp_data.overlay_size

The size of the returned fp_data structure, and therefore the offset to the start of the next entry in the data buffer.

fp_data.ms_appl_name

Name of the currently active focal point application. This is either one of the MS Discipline-Specific Application Programs specified in the *Systems Network Architecture: Management Services Reference* (see "Related Publications"), or an EBCDIC string, using type-1134 characters, padded on the right with spaces if the name is shorter than 8 characters.

fp_data.ms_category

Management Services category. This is either one of the category names specified in the *Systems Network Architecture: Management Services Reference* (see "Related Publications"), or an EBCDIC string, using type-1134 characters, padded on the right with spaces if the name is shorter than 8 characters.

fp_data.description

A null-terminated text string describing the focal point, as specified in the definition of the focal point.

fp_data.fp_fqcp_name

Fully qualified control point name of the currently active focal point. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

fp_data.bkup_appl_name

Backup focal point application name. This is either one of the MS Discipline-Specific Application Programs specified in the *Systems Network Architecture: Management Services Reference* (see "Related Publications"), or an EBCDIC string, using type-1134 characters, padded on the right with spaces if the name is shorter than 8 characters.

fp_data.bkup_fp_fqcp_name

Fully qualified control point name of the backup focal point. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

fp_data.implicit_appl_name

Name of the implicit focal point application (specified using DEFINE_FOCAL_POINT). This is either one of the MS Discipline-Specific Application Programs specified in the *Systems Network Architecture: Management Services Reference* (see "Related Publications"), or an EBCDIC string, using type-1134 characters, padded on the right with spaces if the name is shorter than 8 characters.

fp_data.implicit_fp_fqcp_name

Fully qualified control point name of the implicit focal point (specified using DEFINE_FOCAL_POINT). This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

fp_data.fp_type

Type of focal point. Refer to the IBM *Systems Network Architecture: Management Services Reference* (see "Related Publications") for further detail. This is one of the following:

AP_EXPLICIT_PRIMARY_FP

AP_IMPLICIT_PRIMARY_FP

AP_BACKUP_FP

AP_DEFAULT_PRIMARY_FP

AP_DOMAIN_FP

AP_HOST_FP

AP_NO_FP

fp_data.fp_status

Status of the focal point. This is one of the following:

AP_ACTIVE

The focal point is currently active.

AP_NOT_ACTIVE

The focal point is currently not active.

AP_PENDING

The focal point is pending active. This occurs after an implicit request has been sent to the focal point and before the response has been received.

AP_NEVER_ACTIVE

No focal point information is available for the specified category although application registrations for the category have been accepted.

fp_data.fp_routing

Specifies whether applications should use default or direct routing to route traffic to the focal point. This is one of the following:

AP_DEFAULT

The MDS_MU should be delivered to the focal point using default routing.

AP_DIRECT

The MDS_MU should be routed on a session directly to the focal point.

fp_data.number_of_appls

Number of applications registered for this focal point category.

appl_name

Name of application registered for focal point category. This is either one of the MS Discipline-Specific Application Programs specified in the *Systems Network Architecture: Management Services Reference* (see "Related Publications"), or an EBCDIC string, using type-1134 characters, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_MS_CATEGORY

The *list_options* parameter was set to AP_LIST_INCLUSIVE, to list all entries starting from the supplied name, but the *ms_category* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node

configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_GLOBAL_LOG_TYPE

This verb allows a NOF application to determine the types of information that SNAplus2 records in log files. It specifies default values that are used on all servers (unless they are overridden on a particular server by SET_LOG_TYPE); QUERY_LOG_TYPE can be used to determine the values being used on a particular server.

SNAplus2 logs messages for the following types of event:

Problem	An abnormal event that degrades the system in a way perceptible to a user (such as abnormal termination of a session).
Exception	An abnormal event that may degrade the system but that is not immediately perceptible to a user (such as receiving a message that is not valid from the remote system).
Audit	A normal event (such as starting a session).

Problem and exception messages are logged to the error log file; audit messages are logged to the audit log file. Problem messages are always logged and cannot be disabled, but you can specify whether to log each of the other two message types. For each of the two files (audit and error), you can specify whether to use succinct logging (including only the text of the message and a summary of the message source) or full logging (including full details of the message source, cause, and any action required).

This verb must be issued to the node currently acting as the central logger; for information about accessing this node, see "CONNECT_NODE".

VCB Structure

```
typedef struct query_global_log_type
{
   AP_UINT16       opcode;           /* verb operation code              */
   unsigned char   reserv2;          /* reserved                         */
   unsigned char   format;           /* reserved                         */
   AP_UINT16       primary_rc;       /* primary return code              */
   AP_UINT32       secondary_rc;     /* secondary return code            */
   unsigned char   audit;            /* audit logging on or off          */
```

```
   unsigned char   exception;         /* exception logging on or off         */
   unsigned char   succinct_audits;   /* use succinct logging in audit file? */
   unsigned char   succinct_errors;   /* use succinct logging in error file? */
} QUERY_GLOBAL_LOG_TYPE;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_GLOBAL_LOG_TYPE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

audit

This parameter indicates whether audit messages are recorded. Possible values are:

AP_YES

Audit messages are recorded.

AP_NO

Audit messages are not recorded.

exception

This parameter indicates whether exception messages are recorded. Possible values are:

AP_YES

Exception messages are recorded.

AP_NO

Exception messages are not recorded.

succinct_audits

This parameter indicates whether succinct logging or full logging is used in the audit log file. Possible values are:

AP_YES

Succinct logging: each message in the log file contains a summary of the message header information (such as the message number, log type, and system name) and the message text string and parameters. To obtain more details of the cause of the log and any action required, you can use the snaphelp utility.

AP_NO

Full logging: each message in the log file includes a full listing of the message header information, the message text string and parameters, and additional information about the cause of the log and any action required.

If you are using central logging, the choice of succinct or full logging for messages from all computers is determined by the setting of this parameter on the server acting as the central logger; this setting may either be from the SET_GLOBAL_LOG_TYPE verb, or from a SET_LOG_TYPE verb issued to that server to override the default.

succinct_errors

This parameter indicates whether succinct logging or full logging is used in the error log file; this applies to both exception logs and problem logs. The possible values and their meanings are the same as for the *succinct_audits* parameter.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_NOT_CENTRAL_LOGGER

The verb was issued to a node that is not the central logger.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_KERNEL_MEMORY_LIMIT

This verb returns information about the amount of kernel memory that SNAplus2 is currently using, the maximum amount it has used, and the configured limit. This allows you to check memory usage and set the limit appropriately, to ensure that sufficient memory is available for SNAplus2 components and for other programs on the HP-UX computer.

You can specify the kernel memory limit when starting the SNAplus2 software (for more information, see the *HP-UX SNAplus2 Administration Guide*), or modify it later when the node is running (using the SET_KERNEL_MEMORY_LIMIT verb).

VCB Structure

```
typedef struct query_kernel_memory_limit
{
   AP_UINT16        opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   AP_UINT16        primary_rc;       /* primary return code               */
   AP_UINT32        secondary_rc;     /* secondary return code             */
   AP_UINT32        limit;            /* kernel memory limit, 0 => no limit */
   AP_UINT32        actual;           /* current amount of memory allocated */
   AP_UINT32        max_used;         /* maximum amount of memory allocated */
   unsigned char    reset_max_used;   /* set max_used = actual             */
} QUERY_KERNEL_MEMORY_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_KERNEL_MEMORY_LIMIT

reset_max_used

Specify whether SNAplus2 should reset the *max_used* value (after returning it on this verb) to match the amount of memory currently allocated. This ensures that a subsequent QUERY_KERNEL_MEMORY_LIMIT verb will return

the maximum amount used since this verb, rather than the maximum amount used since the system was started (or since the *max_used* value was last reset). Possible values are:

AP_YES

Reset the *max_used* value to match the current memory allocation.

AP_NO

Do not reset the *max_used* value.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

limit

The maximum amount of kernel memory, in bytes, that SNAplus2 is permitted to use at any time. If a SNAplus2 component attempts to allocate kernel memory that would take the total amount of memory currently allocated to SNAplus2 components above this limit, the allocation attempt will fail. A value of zero indicates no limit.

actual

The amount of kernel memory, in bytes, currently allocated to SNAplus2 components.

max_used

The maximum amount of kernel memory, in bytes, that has been allocated to SNAplus2 components at any time since the *max_used* parameter was last reset (as described for *reset_max_used* above), or since the SNAplus2 software was started.

reset_max_used

Specifies whether SNAplus2 resets the *max_used* value (after returning it on this command) to match the amount of memory currently allocated. This ensures that a subsequent QUERY_KERNEL_MEMORY_LIMIT verb will return the maximum amount used since this command was issued, rather than the maximum amount used since the system was started (or since the *max_used* value was last reset). Possible values are:

AP_YES

SNAplus2 resets the *max_used* value to match the current memory allocation.

AP_NO

SNAplus2 does not reset the *max_used* value.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LOCAL_LU

QUERY_LOCAL_LU returns information about local LUs.

This verb can be used to obtain either summary or detailed information, about a specific LU or about multiple LUs, depending on the options used. It can also obtain information about the LU associated with the CP (the default LU).

VCB Structure

```
typedef struct query_local_lu
{
   AP_UINT16          opcode;               /* verb operation code          */
   unsigned char      reserv2;              /* reserved                     */
   unsigned char      format;               /* reserved                     */
   AP_UINT16          primary_rc;           /* primary return code          */
   AP_UINT32          secondary_rc;         /* secondary return code        */
   unsigned char      *buf_ptr;             /* pointer to buffer            */
   AP_UINT32          buf_size;             /* buffer size                  */
   AP_UINT32          total_buf_size;       /* total buffer size required   */
   AP_UINT16          num_entries;          /* number of entries            */
   AP_UINT16          total_num_entries;    /* total number of entries      */
   unsigned char      list_options;         /* listing options              */
   unsigned char      reserv3;              /* reserved                     */
   unsigned char      lu_name[8];           /* LU name                      */
   unsigned char      lu_alias[8];          /* LU alias                     */
   unsigned char      pu_name[8];           /* PU name filter               */
} QUERY_LOCAL_LU;


typedef struct local_lu_summary
{
   AP_UINT16          overlay_size;         /* size of returned entry       */
   unsigned char      lu_name[8];           /* LU name                      */
   unsigned char      lu_alias[8];          /* LU alias                     */
   unsigned char      description[32];      /* resource description         */
   unsigned char      reserv1[16];          /* reserved                     */
} LOCAL_LU_SUMMARY;


typedef struct local_lu_detail
{
   AP_UINT16          overlay_size;         /* size of returned entry       */
```

```
   unsigned char      lu_name[8];            /* LU name                      */
   LOCAL_LU_DEF_DATA  def_data;              /* defined data                 */
   LOCAL_LU_DET_DATA  det_data;              /* determined data              */
} LOCAL_LU_DETAIL;


typedef struct local_lu_def_data
{
   unsigned char      description[32];       /* resource description         */
   unsigned char      reserv1;               /* reserved                     */
   unsigned char      security_list_name[14] /* security access list name    */
   unsigned char      lu_alias[8];           /* local LU alias               */
   unsigned char      nau_address;           /* NAU address                  */
   unsigned char      syncpt_support;        /* is Syncpoint supported?      */
   AP_UINT16          lu_session_limit;      /* LU session limit             */
   unsigned char      default_pool;          /* is LU in the pool of default */
                                             /*  LUs?                        */
   unsigned char      reserv2;               /* reserved                     */
   unsigned char      pu_name[8];            /* PU name                      */
   unsigned char      lu_attributes;         /* LU attributes                */
   unsigned char      sscp_id[6];            /* SSCP ID                      */
   unsigned char      reserv3[1];            /* reserved                     */
   ATTACH_ROUTING_DATA attach_routing_data;  /* routing data for incoming    */
                                             /*  attaches                    */
} LOCAL_LU_DEF_DATA;


typedef struct local_lu_det_data
{
   unsigned char      lu_sscp_sess_active;   /* Is LU-SSCP session active    */
   unsigned char      appl_conn_active;      /* application is using LU      */
   unsigned char      reserve1[2];           /* reserved                     */
   SESSION_STATS      lu_sscp_stats;         /* LU-SSCP session statistics   */
   unsigned char      sscp_id[6];            /* SSCP ID                      */
} LOCAL_LU_DET_DATA;


typedef struct session_stats
   AP_UINT16          rcv_ru_size;           /* session receive RU size      */
   AP_UINT16          send_ru_size;          /* session send Ru size         */
   AP_UINT16          max_send_btu_size;     /* max send BTU size            */
   AP_UINT16          max_rcv_btu_size;      /* max rcv BTU size             */
   AP_UINT16          max_send_pac_win;      /* max send pacing window size  */
   AP_UINT16          cur_send_pac_win;      /* current send pacing win size */
   AP_UINT16          max_rcv_pac_win;       /* max receive pacing win size  */
   AP_UINT16          cur_rcv_pac_win;       /* current receive pacing       */
                                             /*  window size                 */
```

```
   NB_COUNTER         send_data_frames;      /* number of data frames sent   */
   NB_COUNTER         send_fmd_data_frames;  /* num of fmd data frames sent  */
   NB_COUNTER         send_data_bytes;       /* number of data bytes sent    */
   NB_COUNTER         rcv_data_frames;       /* num data frames received     */
   NB_COUNTER         rcv_fmd_data_frames;   /* num of fmd data frames recvd */
   NB_COUNTER         rcv_data_bytes;        /* number of data bytes received*/
   unsigned char      sidh;                  /* session ID high byte         */
   unsigned char      sidl;                  /* session ID low byte          */
   unsigned char      odai;                  /* ODAI bit set                 */
   unsigned char      ls_name;               /* link station name            */
   unsigned char      pacing_type;           /* type of pacing in use        */
} SESSION_STATS;

typedef struct routing_data
{
   unsigned char      sys_name[64];          /* Name of target system for TP */
   signed long        timeout;               /* timeout value in seconds     */
   unsigned char      back_level;            /* is target system back-level? */
   unsigned char      reserved[59];          /* reserved                     */
} ROUTING_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_LOCAL_LU

overlay_size For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size Size of the supplied data buffer.

num_entries Maximum number of LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list from which SNAplus2 should

begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *lu_name* or *lu_alias* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *lu_name* or *lu_alias* parameter.

If AP_FIRST_IN_LIST is specified, you can also include the following option, using a logical OR operation:

AP_LIST_BY_ALIAS

The list is returned in order of LU alias rather than LU name. This option is only valid if AP_FIRST_IN_LIST is also specified. (For AP_LIST_FROM_NEXT or AP_LIST_INCLUSIVE, the list is in order of LU alias or LU name, depending on which was specified as the index into the list.)

For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs". The list is in EBCDIC lexicographical order (irrespective of the length of each name).

lu_name — Fully qualified name of the LU for which information is required, or the name to be used as an index into the

list of LUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. To identify the LU by its alias instead of its name, set this parameter to 8 binary zeros, and specify the alias in the *lu_alias* parameter; to identify the default LU, set both *lu_name* and *lu_alias* to 8 binary zeros.

The name is an 8-byte EBCDIC string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

lu_alias LU alias of the LU for which information is required, or the name to be used as an index into the list of LUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters. To identify the LU by its LU name instead of its alias, set this parameter to 8 binary zeros, and specify the name in the *lu_name* parameter; to identify the default LU, set both *lu_name* and *lu_alias* to 8 binary zeros.

pu_name PU name filter. To return information only on LUs associated with a specific PU, specify the PU name; to return information without filtering on PU name, set this parameter to 8 binary zeros.

The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

local_lu_summary.overlay_size

The size of the returned local_lu_summary structure, and therefore the offset to the start of the next entry in the data buffer.

local_lu_summary.lu_name

LU name. This name is an 8-byte type-A EBCDIC character string.

local_lu_summary.lu_alias

LU alias. This is an 8-byte ASCII character string.

local_lu_summary.description

A null-terminated text string describing the local LU, as specified in the definition of the LU.

local_lu_detail.overlay_size

The size of the returned local_lu_detail structure, and therefore the offset to the start of the next entry in the data buffer.

local_lu_detail.lu_name

LU name. This name is an 8-byte type-A EBCDIC character string.

local_lu_detail.def_data.description

A null-terminated text string describing the local LU,

as specified in the definition of the LU.

local_lu_detail.def_data.security_list_name

Name of the security access list used by the local LU (defined using the DEFINE_SECURITY_ACCESS_LIST verb). If this parameter is set to 14 binary zeros, the LU is available for use by any user.

local_lu_detail.def_data.lu_alias

LU alias. This is an 8-byte ASCII character string.

local_lu_detail.def_data.nau_address

Network accessible unit address of the LU. This is in the range 1-255 if the LU is a dependent LU, or zero if the LU is an independent LU.

local_lu_detail.def_data.syncpt_support

Specifies whether the LU supports Syncpoint functions. Possible values are:

AP_YES

Syncpoint is supported.

AP_NO

Syncpoint is not supported.

local_lu_detail.def_data.lu_session_limit

Maximum total number of sessions (across all modes) for the local LU. A value of zero indicates that there is no limit.

local_lu_detail.def_data.default_pool

Specifies whether the LU is in the pool of default dependent LUs. When an application attempts to start a conversation without specifying a local LU name, SNAplus2 will select an unused LU from this pool. Possible values are:

AP_YES

The LU is in the pool of default LUs, and can be used by applications that do not specify an LU name.

AP_NO

The LU is not in the pool.

If the LU is an independent LU, this parameter is reserved.

local_lu_detail.def_data.pu_name

For dependent LUs, this parameter identifies the PU that this LU will use. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if necessary. For independent LUs, this field is not used; it is set to 8 binary zeros.

local_lu_detail.def_data.lu_attributes

Configured LU attributes. Possible values are:

AP_NONE

No additional information identified.

AP_DISABLE_PWSUB

Password substitution support is disabled for the local LU.

local_lu_detail.def_data.sscp_id

Specifies the ID of the SSCP permitted to activate this LU. It is a 6-byte binary field. This parameter is used only by dependent LUs, and is set to all binary zeros for independent LUs or if the LU can be activated by any SSCP.

local_lu_detail.def_data.attach_routing_data.sys_name

The name of the target computer for incoming Allocate requests (requests from a partner TP to start an APPC or CPI-C conversation) that arrive at this local LU. This identifies the computer where the target TP runs.

If this parameter is set to binary zeros, SNAplus2 routes the incoming Allocate request dynamically to a running copy of the TP, if available, or attempts to start the TP on the same computer as the local LU.

local_lu_detail.def_data.attach_routing_data.timeout

The timeout value (in seconds) for dynamic load

requests. A request will time out if the invoked TP has not issued a Receive_Allocate verb (APPC), or Accept_Conversation or Accept_Incoming (CPI-C), within this time. A value of -1 indicates no timeout (dynamic load requests will wait indefinitely).

local_lu_detail.def_data.attach_routing_data.back_level

Specifies whether the target computer specified by the *sys_name* parameter above is a back-level computer. This parameter is used only when you are in the process of migrating a client-server SNAplus2 system to a later version, so that not all servers and clients on the system are running the same version; it is reserved otherwise. Possible values are:

AP_YES

The node that owns this LU is running the newer version of SNAplus2, but the target system is running the older version.

AP_NO

Both the node that owns this LU and the target system are running the same version of SNAplus2.

local_lu_detail.det_data.lu_sscp_session_active

Specifies whether the LU-SSCP session is active. If *def_data.nau_address* is zero, this parameter is reserved. Possible values are:

AP_YES

The LU-SSCP session is active.

AP_NO

The LU-SSCP session is not active.

local_lu_detail.det_data.appl_conn_active

Specifies whether an application is using the LU. If *def_data.nau_address* is zero, this parameter is reserved. Possible values are:

AP_YES

An application is using the LU.

AP_NO

No application is using the LU.

local_lu_detail.det_data.lu_sscp_stats.rcv_ru_size

This parameter is always reserved.

local_lu_detail.det_data.lu_sscp_stats.send_ru_size

This parameter is always reserved.

local_lu_detail.det_data.lu_sscp_stats.max_send_btu_size

Maximum basic transmission unit (BTU) size that can be sent.

local_lu_detail.det_data.lu_sscp_stats.max_rcv_btu_size

Maximum BTU size that can be received.

local_lu_detail.det_data.lu_sscp_stats.max_send_pac_win

This parameter is always set to zero.

local_lu_detail.det_data.lu_sscp_stats.cur_send_pac_win

This parameter is always set to zero.

local_lu_detail.det_data.lu_sscp_stats.max_rcv_pac_win

This parameter is always set to zero.

local_lu_detail.det_data.lu_sscp_stats.cur_rcv_pac_win

This parameter is always set to zero.

local_lu_detail.det_data.lu_sscp_stats.send_data_frames

Number of normal flow data frames sent

local_lu_detail.det_data.lu_sscp_stats.send_fmd_data_frames

Number of normal flow function management data (FMD) frames sent.

local_lu_detail.det_data.lu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

local_lu_detail.det_data.lu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

local_lu_detail.det_data.lu_sscp_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

local_lu_detail.det_data.lu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

local_lu_detail.det_data.lu_sscp_stats.sidh

Session ID high byte.

local_lu_detail.det_data.lu_sscp_stats.sidl

Session ID low byte.

local_lu_detail.det_data.lu_sscp_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the ACTLU sets this parameter to zero if the local node contains the primary link station, and sets it to one if the ACTLU sender is the node containing the secondary link station.

local_lu_detail.det_data.lu_sscp_stats.ls_name

Link station name associated with the statistics This is an 8-byte string in a locally displayable character set. All eight bytes are significant. This parameter can be used to correlate this session with the link over which the session flows.

local_lu_detail.det_data.lu_sscp_stats.pacing_type

Receive pacing type in use on the LU-SSCP session. This parameter is set to AP_NONE.

local_lu_detail.det_data.sscp_id

This parameter is a 6-byte field containing the SSCP ID received in the ACTPU for the PU used by this LU.

This parameter is used only by dependent LUs, and is set to all binary zeros for independent LUs or if *lu_sscp_sess_active* is not set to AP_YES.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_ALIAS

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_alias* parameter was not valid.

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LOG_FILE

This verb allows the application to determine the name of the file that SNAplus2 uses to record audit or error log messages, the name of the backup log file, and the file size at which log information is copied to the backup file.

VCB Structure

```
typedef struct query_log_file
{
   AP_UINT16         opcode;                  /* verb operation code     */
   unsigned char     reserv2;                 /* reserved                */
   unsigned char     format;                  /* reserved                */
   AP_UINT16         primary_rc;              /* primary return code     */
   AP_UINT32         secondary_rc;            /* secondary return code   */
   unsigned char     log_file_type;           /* type of log file        */
   unsigned char     file_name[81];           /* file name               */
   unsigned char     backup_file_name[81];    /* backup file name        */
   AP_UINT32         file_size;               /* log file size           */
} QUERY_LOG_FILE;
```

Supplied Parameters

opcode AP_QUERY_LOG_FILE

log_file_type The type of log file being queried. Possible values are:

AP_AUDIT

Audit log file (audit messages only).

AP_ERROR

Error log file (problem and exception messages).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

file_name

Name of the log file. This parameter is an ASCII string of 1-80 characters, followed by a null (0x00) character.

If no path is included, the file is stored in the default directory for diagnostics files, /var/opt/sna; if a path is included, this is either a full path (starting with a / character) or the path relative to the default directory.

backup_file_name

Name of the backup log file. This parameter is an ASCII string of 1-80 characters, followed by a null (0x00) character.

When the log file reaches the size specified by *file_size* below, SNAplus2 copies the current contents of the log file to this file and then clears the log file. You can also request a backup at any time using the SET_LOG_FILE verb.

If no path is included, the file is stored in the default directory for diagnostics files, /var/opt/sna; if a path is included, this is either a full path (starting with a / character) or the path relative to the default directory.

file_size

The maximum size of the log file specified by *log_file_type*. When a message written to the file causes the file size to exceed this limit, SNAplus2 clears the backup log file, copies the current contents of the log file to the backup log file, and then clears the log file. This means that the maximum amount of disk space taken up by log files is approximately twice the value of *file_size*.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_FILE_TYPE The *log_file_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LOG_TYPE

This verb allows a NOF application to determine the types of information that SNAplus2 records in log files on a particular server, and whether these are the default settings specified on SET_GLOBAL_LOG_TYPE or local settings specified by a previous SET_LOG_TYPE verb.

SNAplus2 logs messages for the following types of event:

Problem	An abnormal event that degrades the system in a way perceptible to a user (such as abnormal termination of a session).
Exception	An abnormal event that may degrade the system but that is not immediately perceptible to a user (such as receiving a message that is not valid from the remote system).
Audit	A normal event (such as starting a session).

Problem and exception messages are logged to the error log file; audit messages are logged to the audit log file. Problem messages are always logged and cannot be disabled, but you can specify whether to log each of the other two message types. For each of the two files (audit and error), you can specify whether to use succinct logging (including only the text of the message and a summary of the message source) or full logging (including full details of the message source, cause, and any action required).

VCB Structure

```
typedef struct query_log_type
{
   AP_UINT16       opcode;           /* verb operation code                */
   unsigned char   reserv2;          /* reserved                           */
   unsigned char   format;           /* reserved                           */
   AP_UINT16       primary_rc;       /* primary return code                */
   AP_UINT32       secondary_rc;     /* secondary return code              */
   unsigned char   override;         /* overriding global settings?        */
   unsigned char   audit;            /* audit logging on or off            */
   unsigned char   exception;        /* exception logging on or off        */
   unsigned char   succinct_audits;  /* use succinct logging in audit file? */
   unsigned char   succinct_errors;  /* use succinct logging in error file? */
```

```
} QUERY_LOG_TYPE;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_LOG_TYPE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

override

Specifies whether the log types and succinct or full logging options returned on this verb are the global log types specified on SET_GLOBAL_LOG_TYPE, or local values specified on SET_LOG_TYPE. Possible values are:

AP_YES

The *audit*, *exception*, and *succinct_** parameters returned are local settings overriding the global settings.

AP_NO

The *audit*, *exception*, and *succinct_** parameters returned are the global settings, which are not being overridden.

audit

This parameter indicates whether audit messages are recorded. Possible values are:

AP_YES

Audit messages are recorded.

AP_NO

Audit messages are not recorded.

exception

This parameter indicates whether exception messages are recorded. Possible values are:

AP_YES

Exception messages are recorded.

AP_NO

Exception messages are not recorded.

succinct_audits

This parameter indicates whether succinct logging or full logging is used in the audit log file. Possible values are:

AP_YES

Succinct logging: each message in the log file contains a summary of the message header information (such as the message number, log type, and system name) and the message text string and parameters. To obtain more details of the cause of the log and any action required, you can use the snaphelp utility.

AP_NO

Full logging: each message in the log file includes a full listing of the message header information, the message text string and parameters, and additional information about the cause of the log and any action required.

If you are using central logging, the choice of succinct or full logging for messages from all computers is determined by the setting of this parameter on the server acting as the central logger; this setting may either be from the SET_GLOBAL_LOG_TYPE verb, or from a SET_LOG_TYPE verb issued to that server to override the default.

succinct_errors

This parameter indicates whether succinct logging or

full logging is used in the error log file; this applies to both exception logs and problem logs. The possible values and their meanings are the same as for the *succinct_audits* parameter.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LS

QUERY_LS returns a list of information about the link stations defined at the node. This information is structured as “determined data” (data gathered dynamically during execution, returned only if the node is active) and “defined data” (data supplied on DEFINE_LS).

This verb can be used to obtain either summary or detailed information, about a specific LS or about multiple LSs, depending on the options used.

VCB Structure

```
typedef struct query_ls
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* Primary return code          */
   AP_UINT32        secondary_rc;          /* Secondary return code        */
   unsigned char    *buf_ptr;              /* pointer to buffer            */
   AP_UINT32        buf_size;              /* buffer size                  */
   AP_UINT32        total_buf_size;        /* total buffer size required   */
   AP_UINT16        num_entries;           /* number of entries            */
   AP_UINT16        total_num_entries;     /* total number of entries      */
   unsigned char    list_options;          /* listing options              */
   unsigned char    reserv3;               /* reserved                     */
   unsigned char    ls_name[8];            /* name of link station         */
   unsigned char    port_name[8];          /* port used by link station    */
} QUERY_LS;

typedef struct ls_summary
{
   AP_UINT16        overlay_size;          /* size of returned entry       */
   unsigned char    ls_name[8];            /* link station name            */
   unsigned char    description[32];        /* resource description        */
   unsigned char    reserv1[16];           /* reserved                     */
   unsigned char    dlc_type;              /* DLC type                     */
   unsigned char    state;                 /* link station state           */
   AP_UINT16        act_sess_count;        /* currently active sessions    */
                                           /*  count                       */
   unsigned char    det_adj_cp_name[17];   /* determined adjacent CP name  */
   unsigned char    det_adj_cp_type;       /* determined adjacent node type*/
   unsigned char    port_name[8];          /* port name                    */
```

```
   unsigned char    adj_cp_name[17];         /* adjacent CP name             */
   unsigned char    adj_cp_type;             /* adjacent node type           */
} LS_SUMMARY;

typedef struct ls_detail
{
   AP_UINT16        overlay_size;            /* size of returned entry       */
   unsigned char    ls_name[8];              /* link station name            */
   LS_DET_DATA      det_data;                /* determined data              */
   LS_DEF_DATA      def_data;                /* defined data                 */
} LS_DETAIL;

typedef struct ls_det_data
{
   AP_UINT16        act_sess_count;          /* currently active sessions    */
                                             /*  count                       */
   unsigned char    dlc_type;                /* DLC type                     */
   unsigned char    state;                   /* link station state           */
   unsigned char    sub_state;               /* link station sub state       */
   unsigned char    det_adj_cp_name[17];     /* adjacent CP name             */
   unsigned char    det_adj_cp_type;         /* adjacent node type           */
   unsigned char    dlc_name[8];             /* name of DLC                  */
   unsigned char    dynamic;                 /* specifies whether LS is      */
                                             /*  dynamic                     */
   unsigned char    migration;               /* supports migration partners  */
   unsigned char    tg_num;                  /* TG number                    */
   LS_STATS         ls_stats;                /* link station statistics      */
   AP_UINT32        start_time;              /* time LS started              */
   AP_UINT32        stop_time;               /* time LS stopped              */
   AP_UINT32        up_time;                 /* total time LS active         */
   AP_UINT32        current_state_time;      /* time in current state        */
   unsigned char    deact_cause;             /* deactivation cause           */
   unsigned char    hpr_support;             /* reserved                     */
   unsigned char    anr_label[2];            /* reserved                     */
   unsigned char    hpr_link_lvl_error;      /* reserved                     */
   unsigned char    auto_act;                /* auto-activation supported    */
   unsigned char    ls_role;                 /* LS role                      */
   unsigned char    reserva;                 /* reserved                     */
   unsigned char    node_id[4];              /* determined node ID           */
   AP_UINT16        active_isr_count;        /* active isr count             */
   AP_UINT16        active_lu_sess_count;    /* count of active LU sessions  */
   AP_UINT16        active_sscp_sess_count;  /* count of active SSCP sessions*/
   ANR_LABEL        reverse_anr_label;       /* reserved                     */
   LINK_ADDRESS     local_address;           /* Local address                */
   AP_UINT16        max_send_btu_size;       /* Max send BTU size            */
```

```
   unsigned char    reservb[66];            /* reserved                    */
} LS_DET_DATA;


typedef struct ls_def_data
{
   unsigned char    description[32];        /* resource description        */
   unsigned char    initially_active;       /* is this LS initially active? */
   AP_UINT16        reserv2;                /* reserved                    */
   AP_UINT16        react_timer;            /* timer for retrying failed LS */
   AP_UINT16        react_timer_retry;      /* retry count for failed LS   */
   unsigned char    reserv3[10];            /* reserved                    */
   unsigned char    port_name[8];           /* name of associated port     */
   unsigned char    adj_cp_name[17];        /* adjacent CP name            */
   unsigned char    adj_cp_type;            /* adjacent node type          */
   LINK_ADDRESS     dest_address;           /* destination address         */
   unsigned char    auto_act_supp;          /* auto-activate supported     */
   unsigned char    tg_number;              /* pre-assigned TG number      */
   unsigned char    limited_resource;       /* limited resource            */
   unsigned char    solicit_sscp_sessions;  /* solicit SSCP sessions       */
   unsigned char    pu_name[8];             /* Local PU name (reserved if  */
                                            /*  solicit_sscp_sessions is   */
                                            /*  set to AP_NO)              */
   unsigned char    disable_remote_act;     /* disable remote activation   */
   unsigned char    dspu_services;          /* Services provided for
                                               downstream PU               */
   unsigned char    dspu_name[8];           /* Downstream PU name (reserved
                                               if dspu_services is AP_NONE)*/
   unsigned char    dlus_name[17];          /* DLUS name if dspu_services  */
                                            /*  is AP_DLUR                 */
   unsigned char    bkup_dlus_name[17];     /* Backup DLUS name if         */
                                            /*  dspu_services is AP_DLUR   */
   unsigned char    hpr_supported;          /* reserved                    */
   unsigned char    hpr_link_lvl_error;     /* reserved                    */
   AP_UINT16        link_deact_timer;       /* deact timer for limited     */
                                            /*  resource                   */
   unsigned char    reserv1;                /* reserved                    */
   unsigned char    default_nn_server;      /* default LS to NN server?    */
   unsigned char    ls_attributes[4];       /* LS attributes               */
   unsigned char    adj_node_id[4];         /* adjacent node ID            */
   unsigned char    local_node_id[4];       /* local node ID               */
   unsigned char    cp_cp_sess_support;     /* CP-CP session support       */
   unsigned char    use_default_tg_chars;   /* Use default tg_chars        */
   TG_DEFINED_CHARS tg_chars;               /* TG characteristics          */
   AP_UINT16        target_pacing_count;    /* target pacing count         */
   AP_UINT16        max_send_btu_size;      /* maximum send BTU size       */
   AP_UINT16        ls_role;                /* link station role           */
```

```
   unsigned char    max_ifrm_rcvd;          /* no. before acknowledgment    */
   AP_UINT16        dlus_retry_timeout;     /* seconds to recontact a DLUS  */
   AP_UINT16        dlus_retry_limit;       /* attempts to recontact a DLUS */
   AP_UINT16        reserv4[30];            /* reserved                     */
   AP_UINT16        link_spec_data_len;     /* length of link specific data */
} LS_DEF_DATA;

typedef struct link_address
{
   AP_UINT16        reserve1;               /* reserved                     */
   AP_UINT16        length;                 /* length                       */
   unsigned char    address[32];            /* address                      */
} LINK_ADDRESS;
```

For Token Ring, Ethernet, or FDDI, the *address* parameter in the link_address structure is replaced by the following:

```
typedef struct tr_address
{
   unsigned char    mac_address[6];         /* MAC address                  */
   unsigned char    lsap_address;           /* local SAP address            */
} TR_ADDRESS;

typedef struct tg_defined_chars
{
   unsigned char    effect_cap;             /* Effective capacity           */
   unsigned char    reserve1[5];            /* Reserved                     */
   unsigned char    connect_cost;           /* Connection Cost              */
   unsigned char    byte_cost;              /* Byte cost                    */
   unsigned char    reserve2;               /* Reserved                     */
   unsigned char    security;               /* Security                     */
   unsigned char    prop_delay;             /* Propagation delay            */
   unsigned char    modem_class;            /* reserved                     */
   unsigned char    user_def_parm_1;        /* User-defined parameter 1     */
   unsigned char    user_def_parm_2;        /* User-defined parameter 2     */
   unsigned char    user_def_parm_3;        /* User-defined parameter 3     */
} TG_DEFINED_CHARS;

typedef struct ls_stats
{
   AP_UINT32       in_xid_bytes;         /* number of XID bytes received     */
   AP_UINT32       in_msg_bytes;         /* number of message bytes received */
   AP_UINT32       in_xid_frames;        /* number of XID frames received    */
   AP_UINT32       in_msg_frames;        /* number of message frames received*/
```

```
   AP_UINT32       out_xid_bytes;        /* number of XID bytes sent         */
   AP_UINT32       out_msg_bytes;        /* number of message bytes sent     */
   AP_UINT32       out_xid_frames;       /* number of XID frames sent        */
   AP_UINT32       out_msg_frames;       /* number of message frames sent    */
   AP_UINT32       in_invalid_sna_frames;       /* number of invalid frames  */
                                        /*  received                        */
   AP_UINT32       in_session_control_frames;   /* number of control frames  */
                                        /*  received                        */
   AP_UINT32       out_session_control_frames;  /* number of control frames  */
                                        /*  sent                            */
   AP_UINT32       echo_rsps;           /* reserved                         */
   AP_UINT32       current_delay;       /* reserved                         */
   AP_UINT32       max_delay;           /* reserved                         */
   AP_UINT32       min_delay;           /* reserved                         */
   AP_UINT32       max_delay_time;      /* reserved                         */
   AP_UINT32       good_xids;           /* successful XID on LS count       */
   AP_UINT32       bad_xids;            /* unsuccessful XID on LS count     */
} LS_STATS;
```

For more details of the link-specific data, see “DEFINE_LS”. The data structure for this data follows the `ls_def_data` structure, but is padded to start on a 4-byte boundary.

Supplied Parameters

The application supplies the following parameters:

opcode	AP_QUERY_LS
overlay_size	For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.
buf_ptr	A pointer to a data buffer that SNAplus2 will use to return the requested information.
buf_size	Size of the supplied data buffer.
num_entries	Maximum number of LSs for which data should be returned. To request data for a specific LS rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *ls_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *ls_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

ls_name Link station name. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

port_name Port name filter. To return information only on LSs associated with a specific port, specify the name of the port. This is an 8-byte ASCII string), padded on the right with spaces if the name is shorter than 8 characters. To return information about all LSs without filtering on the port name, set this parameter to 8 binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

ls_summary.overlay_size

The size of the returned ls_summary structure, and therefore the offset to the start of the next entry in the data buffer.

ls_summary.ls_name

Link station name. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

ls_summary.description

A null-terminated text string describing the LS, as specified in the definition of the LS.

ls_summary.dlc_type

Type of DLC. This is one of the following:

AP_SDLC

SDLC

AP_X25

QLLC

AP_TR

Token Ring

AP_ETHERNET

Ethernet

AP_FDDI

FDDI

ls_summary.state

State of this link station. This is one of the following:

AP_ACTIVE

The LS is active.

AP_NOT_ACTIVE

The LS is not active.

AP_PENDING_ACTIVE

The LS is being activated.

AP_PENDING_INACTIVE

The LS is being deactivated.

AP_PENDING_ACTIVE_BY_LR

The LS has failed (or an attempt to activate it has failed), and SNAplus2 is attempting to reactivate it.

ls_summary.act_sess_count

The total number of active sessions (both endpoint and intermediate) using the link.

ls_summary.det_adj_cp_name

Fully qualified name of the adjacent control point. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8

A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

This name is normally determined during activation; it is null if the LS is inactive. However, for an LS to a back-level LEN node (specified by the *adj_cp_type* parameter on DEFINE_LS), this name is taken from the LS definition and is not determined during activation.

ls_summary.det_adj_cp_type

Type of the adjacent node. This is one of the following:

AP_APPN_NODE

Node type is unknown, or LS is inactive.

AP_END_NODE

End node, or up-level LEN node (one that includes the Network Name CV in its XID3).

AP_NETWORK_NODE

Network node.

AP_VRN

Virtual routing node.

The node type is normally determined during activation; it is null if the LS is inactive. However, for an LS to a back-level LEN node (specified by the *adj_cp_type* parameter on DEFINE_LS), the node type is taken from the LS definition and is not determined during activation.

ls_summary.port_name

Name of the port associated with this link station. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

ls_summary.adj_cp_name

Fully qualified name of the adjacent control point; this parameter is null for an implicit link. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8

A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

ls_summary.adj_cp_type

Type of the adjacent node, determined during link activation. This is one of the following:

AP_APPN_NODE

Node type is unknown, or LS is inactive.

AP_END_NODE

End node, or up-level LEN node (one that includes the Network Name CV in its XID3).

AP_NETWORK_NODE

Network node.

AP_BACK_LEVEL_LEN_NODE

Back-level LEN node (one that does not include the Network Name CV in its XID3).

AP_HOST_XID3

Host node; SNAplus2 should respond to a polling XID from the node with a format 3 XID.

AP_HOST_XID0

Host node; SNAplus2 should respond to a polling XID from the node with a format 0 XID.

AP_DSPU_XID

Downstream PU; SNAplus2 should include XID exchange in link activation. The *dspu_name* and *dspu_services* fields must also be set.

AP_DSPU_NOXID

Downstream PU; SNAplus2 should not include XID exchange in link activation. The *dspu_name* and *dspu_services* fields must also be set.

AP_VRN

Virtual routing node.

ls_detail.overlay_size

The size of the returned `ls_detail` structure, and therefore the offset to the start of the next entry in the data buffer.

`ls_detail.ls_name`

Link station name. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

`ls_detail.det_data.act_sess_count`

The total number of active sessions using the link.

`ls_detail.det_data.dlc_type`

Type of DLC. This is one of the following:

`AP_SDLC`

SDLC

`AP_X25`

QLLC

`AP_TR`

Token Ring

`AP_ETHERNET`

Ethernet

`AP_FDDI`

FDDI

`ls_detail.det_data.state`

State of this link station. This is one of the following:

`AP_ACTIVE`

The LS is active.

`AP_NOT_ACTIVE`

The LS is not active.

`AP_PENDING_ACTIVE`

The LS is being activated.

`AP_PENDING_INACTIVE`

The LS is being deactivated.

AP_PENDING_ACTIVE_BY_LR

The LS has failed (or an attempt to activate it has failed), and SNAplus2 is attempting to reactivate it.

ls_detail.det_data.sub_state

This field provides more detailed information about the state of this link station. Possible values are:

AP_SENT_CONNECT_OUT

AP_PENDING_XID_EXCHANGE

AP_SENT_ACTIVATE_AS

AP_SENT_SET_MODE

AP_ACTIVE

AP_SENT_DEACTIVATE_AS_ORDERLY

AP_SENT_DISCONNECT

AP_WAITING_STATS

AP_RESET

ls_detail.det_data.det_adj_cp_name

Fully qualified name of the adjacent control point. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

This name is normally determined during activation; it is null if the LS is inactive. However, for an LS to a back-level LEN node (specified by the *adj_cp_type* parameter on DEFINE_LS), this name is taken from the LS definition and is not determined during activation.

ls_detail.det_data.det_adj_cp_type

Type of the adjacent node. This is one of the following:

AP_END_NODE

End node.

AP_NETWORK_NODE

Network node.

AP_LEARN_NODE

Node type is unknown.

The node type is normally determined during activation; it is null if the LS is inactive. However, for an LS to a back-level LEN node (specified by the *adj_cp_type* parameter on DEFINE_LS), the node type is taken from the LS definition and is not determined during activation.

ls_detail.det_data.dlc_name

Name of the DLC. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

ls_detail.det_data.dynamic

Specifies whether the link was defined dynamically. Possible values are:

AP_YES

The link was defined dynamically (in response to a connection request from the adjacent node, or to connect dynamically to another node across a Connection Network).

AP_NO

The link was defined explicitly (by DEFINE_LS).

ls_detail.det_data.migration

Specifies whether the adjacent node is a migration level node (such as a Low Entry Networking or LEN node), or a full APPN network node or end node. Possible values are:

AP_YES

The adjacent node is a migration-level node.

AP_NO

The adjacent node is a network node or end node.

AP_UNKNOWN

The adjacent node level is unknown.

ls_detail.det_data.tg_num

Number associated with the TG.

ls_detail.det_data.ls_stats.in_xid_bytes

Total number of XID (Exchange Identification) bytes received on this link station.

ls_detail.det_data.ls_stats.in_msg_bytes

Total number of data bytes received on this link station.

ls_detail.det_data.ls_stats.in_xid_frames

Total number of XID (Exchange Identification) frames received on this link station.

ls_detail.det_data.ls_stats.in_msg_frames

Total number of data frames received on this link station.

ls_detail.det_data.ls_stats.out_xid_bytes

Total number of XID (Exchange Identification) bytes sent on this link station.

ls_detail.det_data.ls_stats.out_msg_bytes

Total number of data bytes sent on this link station.

ls_detail.det_data.ls_stats.out_xid_frames

Total number of XID (Exchange Identification) frames sent on this link station.

ls_detail.det_data.ls_stats.out_msg_frames

Total number of data frames sent on this link station.

ls_detail.det_data.ls_stats.in_invalid_sna_frames

Total number of not valid SNA frames received on this link station.

ls_detail.det_data.ls_stats.in_session_control_frames

Total number of session control frames received on this

link station.

ls_detail.det_data.ls_stats.out_session_control_frames

Total number of session control frames sent on this link station.

ls_detail.det_data.ls_stats.good_xids

Total number of successful XID exchanges that have occurred on this link station since it was started.

ls_detail.det_data.ls_stats.bad_xids

Total number of unsuccessful XID exchanges that have occurred on this link station since it was started.

ls_detail.det_data.start_time

Time since system startup (in hundredths of a second) when the link station was last activated (that is, when the mode setting commands completed).

ls_detail.det_data.stop_time

Time since system startup (in hundredths of a second) when the link station was last deactivated.

ls_detail.det_data.up_time

Total time (in hundredths of a second) that this link station has been active since system startup.

ls_detail.det_data.current_state_time

Total time (in hundredths of a second) that this link station has been in its current state.

ls_detail.det_data.deact_cause

The cause of the last deactivation of the link station. Possible values are:

AP_NONE

The link station has never been deactivated.

AP_DEACT_OPER_ORDERLY

The link station was deactivated as a result of an orderly STOP command from an operator.

AP_DEACT_OPER_IMMEDIATE

The link station was deactivated as a result of an immediate STOP command from an operator.

AP_DEACT_AUTOMATIC

The link station was deactivated automatically, for example because there were no more sessions using the link station.

AP_DEACT_FAILURE

The link station was deactivated because of a failure.

ls_detail.det_data.hpr_support

This parameter is reserved.

ls_detail.det_data.anr_label

This parameter is reserved.

ls_detail.det_data.hpr_link_lvl_error

This parameter is reserved.

ls_detail.det_data.auto_act

Specifies whether the link currently allows remote activation or activation on demand. This is set to AP_NONE if neither is allowed, or to one or both of the following values (combined using a logical OR):

AP_AUTO_ACT

The link can be activated on demand by the local node when a session requires it.

AP_REMOTE_ACT

The link can be activated by the remote node.

ls_detail.det_data.ls_role

The LS role of this link. This is normally taken from the definition of the port that owns the LS (or from the definition of the LS, if this overrides the LS role in the port definition). However, if the LS role is defined to be negotiable, it will be negotiated to either primary or secondary while the LS is active; if this verb is used to query an active LS, the returned LS role is the negotiated role currently in use and not the defined

role. Possible values are:

AP_LS_PRI Primary.

AP_LS_SEC Secondary.

AP_LS_NEG Negotiable.

ls_detail.det_data.node_id

Node ID received from adjacent node during XID exchange. This is a 4-byte hexadecimal string.

ls_detail.det_data.active_isr_count

Number of active intermediate sessions using the link.

ls_detail.det_data.active_lu_sess_count

The count of active LU-LU sessions using this link.

ls_detail.det_data.active_sscp_sess_count

The count of active PU-SSCP sessions using this link.

ls_detail.det_data.reverse_anr_label

This parameter is reserved.

ls_detail.det_data.local_address

The local address of this link station.

ls_detail.det_data.max_send_btu_size

Maximum BTU size that can be sent on this link, as determined by negotiation with the adjacent node. If the link activation has not yet been attempted, a zero value is returned.

ls_detail.def_data.description

A null-terminated text string describing the LS, as specified in the definition of the LS.

ls_detail.def_data.initially_active

Specifies whether this LS is automatically started when the node is started. Possible values are:

AP_YES

The LS is automatically started when the node is started.

AP_NO

The LS is not automatically started; it must be started manually.

ls_detail.def_data.react_timer

Reactivation timer for reactivating a failed LS. If the *react_timer_retry* parameter below is nonzero, to specify that SNAplus2 should retry activating the LS if it fails, this parameter specifies the time in seconds between retries. When the LS fails, or when an attempt to reactivate it fails, SNAplus2 waits for the specified time before retrying the activation. If *react_timer_retry* is zero, this parameter is ignored.

ls_detail.def_data.react_timer_retry

Retry count for reactivating a failed LS. This parameter is used to specify whether SNAplus2 should attempt to reactivate the LS if it fails while in use (or if an attempt to start the LS fails).

Zero indicates that SNAplus2 should not attempt to reactivate the LS; a nonzero value specifies the number of retries to be made. A value of 65,535 indicates that SNAplus2 should retry indefinitely until the LS is activated.

SNAplus2 waits for the time specified by the *react_timer* parameter above between successive retries. If the retry count is reached without successfully reactivating the LS, or if a STOP_LS is issued while SNAplus2 is retrying the activation, no further retries are made; the LS remains inactive unless START_LS is issued for it.

If the *auto_act_supp* parameter is set to AP_YES, the reactivation timer fields are ignored; if the link fails, SNAplus2 does not attempt to reactivate it until the user application that was using the session attempts to restart the session.

ls_detail.def_data.port_name

Name of the port associated with this link station. This is an 8-byte ASCII string, padded on the right with

spaces if the name is shorter than 8 characters. If the link is to a VRN, this field specifies the name of the actual port used to connect to the VRN (as specified in the DEFINE_CN verb).

ls_detail.def_data.adj_cp_name

Fully qualified name of the adjacent control point. This parameter is used only if *adj_cp_type* specifies that the adjacent node is an APPN node or a back-level LEN node.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

ls_detail.def_data.adj_cp_type

Adjacent node type. This is one of the following:

AP_APPN_NODE

APPN-capable node; the node type will be learned during XID exchange.

AP_NETWORK_NODE

Network node.

AP_END_NODE

End node, or up-level LEN node (one that includes the Network Name CV in its XID3).

AP_BACK_LEVEL_LEN_NODE

Back-level LEN node (one that does not include the Network Name CV in its XID3).

AP_HOST_XID3

Host node; SNAplus2 responds to a polling XID from the node with a format 3 XID.

AP_HOST_XID0

Host node; SNAplus2 responds to a polling XID from the node with a format 0 XID.

AP_DSPU_XID

Downstream PU; SNAplus2 includes XID exchange in link activation.

AP_DSPU_NOXID

Downstream PU; SNAplus2 does not include XID exchange in link activation.

For SDLC:

ls_detail.def_data.dest_address

Address of the secondary link station.

The value of this parameter depends on how the port that owns this LS is configured, as follows:

- If the port is used only for incoming calls (*out_link_act_lim* on DEFINE_SDLC_PORT is 0), this parameter is reserved.
- If the port is switched primary and is used for outgoing calls (*port_type* is PORT_SWITCHED, *ls_role* is LS_PRI, and *out_link_act_lim* on DEFINE_SDLC_PORT is a nonzero value), this parameter is set to either 0xFF to accept whatever address is configured at the secondary station, or set it to a 1-byte value in the range 0x01-0xFE (this value must match the value configured at the secondary station).
- For all other port configurations, this parameter is set to a 1-byte value in the range 0x01-0xFE to identify the link station. If the port is primary multi-drop (*ls_role* on DEFINE_SDLC_PORT is LS_PRI and *tot_link_act_lim* is greater than 1), this address must be different for each LS on the port.

For QLLC:

ls_detail.def_data.dest_address

Destination address of link station on the adjacent node. This parameter is used only for SVC outgoing calls (defined by the *vc_type* parameter in the link-specific data, and by the link activation limit

parameters on DEFINE_PORT); it is ignored for incoming calls or for PVC.

For Token Ring, Ethernet, FDDI:

ls_detail.def_data.dest_address.mac_address

MAC address of link station on adjacent node.

If the local and adjacent nodes are on LANs of different types (one Token Ring or FDDI, the other Ethernet) connected by a bridge, you will probably need to reverse the bit order of the bytes in the MAC address. For more information, see "Bit Ordering in MAC Addresses". If the two nodes are on the same LAN, or on LANs of the same type connected by a bridge, no change is required.

ls_detail.def_data.dest_address.lsap_address

Local SAP address of link station on adjacent node.

For all link types:

ls_detail.def_data.auto_act_supp

Specifies whether the link can be activated automatically when required by a session. Possible values are:

AP_YES

The link can be activated automatically.

AP_NO

The link cannot be activated automatically.

ls_detail.def_data.tg_number

Preassigned TG number, used to represent the link when the link is activated. This parameter is used only if the adjacent node is an APPN node (*adj_cp_type* is either AP_NETWORK_NODE or AP_END_NODE); it is ignored otherwise. Zero indicates that the TG number is not preassigned and is negotiated when the link is activated.

ls_detail.def_data.limited_resource

Specifies whether this link station is to be deactivated

when there are no sessions using the link. Possible values are:

AP_NO

The link is not a limited resource and will not be deactivated automatically.

AP_NO_SESSIONS

The link is a limited resource and will be deactivated automatically when no active sessions are using it.

AP_INACTIVITY

The link is a limited resource and will be deactivated automatically when no active sessions are using it, or when no data has flowed on the link for the time period specified by the *link_deact_timer* field.

ls_detail.def_data.solicit_sscp_sessions

Specifies whether to request the adjacent node to initiate sessions between the SSCP and the local CP and dependent LUs. This parameter is used only if the adjacent node is an APPN node (*adj_cp_type* is either AP_NETWORK_NODE or AP_END_NODE); it is ignored otherwise. If the adjacent node is a host (*adj_cp_type* is either AP_HOST_XID3 or AP_HOST_XID0), SNAplus2 always requests the host to initiate SSCP sessions. Possible values are:

AP_YES

Request the adjacent node to initiate SSCP sessions.

AP_NO

Do not request the adjacent node to initiate SSCP sessions.

ls_detail.def_data.pu_name

Name of the local PU that uses this link. This parameter is used only if *adj_cp_type* is set to AP_HOST_XID3 or AP_HOST_XID0, or if *solicit_sscp_sessions* is set to AP_YES; it is reserved otherwise. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded

on the right with EBCDIC spaces.

ls_detail.def_data.disable_remote_act

Specifies whether the LS can be activated by a remote node. Possible values are:

AP_YES

The LS can only be activated by the local node; if the remote node attempts to activate it, SNAplus2 will reject the attempt.

AP_NO

The LS can be activated by the remote node.

ls_detail.def_data.dspu_services

Specifies the services which the local node will provide to the downstream PU across this link. This parameter is used only if the adjacent node is a downstream PU or an APPN node with *solicit_sscp_sessions* set to AP_NO; it is reserved otherwise. Possible values are:

AP_PU_CONCENTRATION

Local node will provide PU concentration for the downstream PU.

AP_DLUR

Local node will provide DLUR services for the downstream PU.

AP_NONE

Local node will provide no services for this downstream PU.

ls_detail.def_data.dspu_name

Name of the downstream PU. This parameter is required if *solicit_sscp_sessions* is set to AP_NO and *dspu_services* is set to AP_PU_CONCENTRATION or AP_DLUR; it is reserved otherwise. The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

ls_detail.def_data.dlus_name

Name of the DLUS node from which DLUR solicits SSCP services when the link to the downstream node is activated. This field is reserved if *dspu_services* is not set to AP_DLUR.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

A string of 17 binary zeros indicates the global default DLUS, defined using the DEFINE_DLUR_DEFAULTS verb. If this parameter is set to zeros and there is no global default DLUS, then DLUR will not initiate SSCP contact when the link is activated.

ls_detail.def_data.bkup_dlus_name

Name of the DLUS node from which DLUR solicits SSCP services when the link to the downstream node is activated. This field is reserved if *dspu_services* is not set to AP_DLUR.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

A string of 17 binary zeros indicates the global backup default DLUS, defined using the DEFINE_DLUR_DEFAULTS verb.

ls_detail.def_data.hpr_supported

This parameter is reserved.

ls_detail.def_data.hpr_link_lvl_error

This parameter is reserved.

ls_detail.def_data.link_deact_timer

Limited resource link deactivation timer (in seconds, minimum 5). If *limited_resource* is set to AP_INACTIVITY, the link will be deactivated if no data flows on it for the time specified by this parameter. A

value of zero indicates no timeout (the link is not deactivated, as though *limited_resource* were set to AP_NO), and that values in the range 1-4 are interpreted as 5.

ls_detail.def_data.default_nn_server

End node: Specifies whether this is a link supporting CP-CP sessions to a network node that can act as the local node's network node server. When the local node has no CP-CP sessions to a network node server and needs to establish them, it checks this parameter on its defined LSs to find a suitable LS to activate. This allows you to specify which adjacent NNs are suitable to act as the NN server (for example, to avoid using NNs that are accessed by expensive or slow links).

Possible values are:

AP_YES

This link supports CP-CP sessions to a network node that can act as the local node's NN server; the local node can automatically activate this link if it needs to contact an NN server.

AP_NO

This link should not be automatically activated in an attempt to contact a network node server.

If the local node is not an end node, this parameter is reserved.

ls_detail.def_data.ls_attributes

This array contains further information about the adjacent node, as described in the following parameters:

ls_detail.def_data.ls_attributes[0]

Host type (normally standard SNA). Possible values are:

AP_SNA

Standard SNA host.

AP_FNA

Fujitsu Network Architecture (VTAM-F) host.

AP_HNA

Hitachi Network Architecture host.

ls_detail.def_data.ls_attributes[1]

Network Name CV suppression for a link to a back-level LEN node.

If *adj_cp_type* is set to AP_BACK_LEVEL_LEN_NODE, this parameter specifies whether to suppress inclusion of the Network Name CV in the format 3 XID sent to the LEN node. Possible values are:

AP_NO

Include the Network Name CV in the XID.

AP_SUPPRESS_CP_NAME

Do not include the Network name CV.

If *adj_cp_type* is set to any other value, this parameter is reserved.

ls_detail.def_data.adj_node_id

Node ID of adjacent node. This is a 4-byte hexadecimal string; a value of 4 zeros indicates that node ID checking is disabled.

ls_detail.def_data.local_node_id

Node ID sent in XIDs on this LS. This is a 4-byte hexadecimal string, consisting of a block number (3 hexadecimal digits) and a node number (5 hexadecimal digits). A value of all zeros indicates that SNAplus2 uses the node ID specified in the DEFINE_NODE verb.

ls_detail.def_data.cp_cp_sess_support

Specifies whether CP-CP sessions are supported. Possible values are:

AP_YES

CP-CP sessions are supported.

AP_NO

CP-CP sessions are not supported.

ls_detail.def_data.use_default_tg_chars

Specifies whether the default TG characteristics supplied on the DEFINE_PORT verb are used. Possible values are:

AP_YES

Use the default TG characteristics; ignore the tg_chars structure on this verb.

AP_NO

Use the tg_chars structure on this verb.

ls_detail.def_data.tg_chars.effect_cap

Actual bits per second rate (line speed). The value is encoded as a 1-byte floating point number, represented by the formula 0.1 mmm * 2 ^{eeeee} where the bit representation of the byte is b'eeeeemmm'. Each unit of effective capacity is equal to 300 bits per second.

ls_detail.def_data.tg_chars.connect_cost

Cost per connect time. Valid values are integer values in the range 0-255, where 0 is the lowest cost per connect time and 255 is the highest.

ls_detail.def_data.tg_chars.byte_cost

Cost per byte. Valid values are integer values in the range 0-255, where 0 is the lowest cost per byte and 255 is the highest.

ls_detail.def_data.tg_chars.security

Security level of the network. Possible values are:

AP_SEC_NONSECURE

No security.

AP_SEC_PUBLIC_SWITCHED_NETWORK

Data is transmitted over a public switched network.

AP_SEC_UNDERGROUND_CABLE

Data is transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT

Data is transmitted over a line in a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT

Data is transmitted over a line in a conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED

Data is encrypted before transmission over the line.

AP_SEC_GUARDED_RADIATION

Data is transmitted over a line that is protected against physical and radiation tapping.

AP_SEC_MAXIMUM

Maximum security.

ls_detail.def_data.tg_chars.prop_delay

Propagation delay: the time that a signal takes to travel the length of the link. Possible values are:

AP_PROP_DELAY_MINIMUM

Minimum propagation delay.

AP_PROP_DELAY_LAN

Delay is less than 480 microseconds (typical for a LAN).

AP_PROP_DELAY_TELEPHONE

Delay is in the range 480-49,512 microseconds (typical for a telephone network).

AP_PROP_DELAY_PKT_SWITCHED_NET

Delay is in the range 49,512-245,760 microseconds (typical for a packet-switched network).

AP_PROP_DELAY_SATELLITE

Delay is greater than 245,760 microseconds (typical for a satellite link).

AP_PROP_DELAY_MAXIMUM

Maximum propagation delay.

ls_detail.def_data.tg_chars.user_def_parm_1 through *def_data.tg_chars.user_def_parm_3*

User-defined parameters, which include other TG characteristics not covered by the above parameters. Each of these parameters is set to a value in the range 0-255.

ls_detail.def_data.target_pacing_count

Numeric value between 1 and 32,767 inclusive indicating the desired pacing window size. (The current version of SNAplus2 does not make use of this value.)

ls_detail.def_data.max_send_btu_size

Maximum BTU size that can be sent.

ls_detail.def_data.ls_role

Link station role. This is normally set to AP_USE_PORT_DEFAULTS, specifying that the LS role is taken from the definition of the port that owns this LS.

If the LS has been defined with a specific LS role overriding the port definition, this is one of the following values:

AP_LS_PRI	Primary
AP_LS_SEC	Secondary
AP_LS_NEG	Negotiable

ls_detail.def_data.max_ifrm_rcvd

Maximum of I-frames that can be received by this link station before an acknowledgment is sent. This value is in the range 0-127. When this field is zero, the value of *max_ifrm_rcvd* from DEFINE_PORT is used as default.

ls_detail.def_data.dlus_retry_timeout

Interval in seconds between second and subsequent attempts to contact the DLUS specified in the *ls_detail.def_data.dlus_name* and *ls_detail.def_data.bkup_dlus_name* parameters.

The interval between the initial attempt and the first retry is always one second. If zero is returned, the default value configured with DEFINE_DLUR_DEFAULTS is used. This parameter is ignored if *ls_detail.def_data.dspu_services* is not set to AP_DLUR.

ls_detail.def_data.dlus_retry_limit

Maximum number of retries after an initial failure to contact the DLUS specified in the *ls_detail.def_data.dlus_name* and *ls_detail.def_data.bkup_dlus_name* parameters. If zero is returned, the default value configured through DEFINE_DLUR_DEFAULTS is used. If 0x0FFFF is returned, SNAplus2 retries indefinitely. This parameter is ignored if *ls_detail.def_data.dspu_services* is not set to AP_DLUR.

ls_detail.def_data.link_spec_data_len

Length of link-specific data that is passed unchanged to link station component during initialization. The data structure for this data follows the ls_def_data structure, but is padded to start on a 4-byte boundary. For more details of the link-specific data, see "DEFINE_LS".

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LINK_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *ls_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LU_0_TO_3

QUERY_LU_0_TO_3 returns information about local LUs of type 0, 1, 2, or 3. This information is structured as “determined data” (data gathered dynamically during execution, returned only if the node is active) and “defined data” (the data supplied by the application on DEFINE_LU_0_TO_3).

This verb can be used to obtain either summary or detailed information, about a specific LU or about multiple LUs, depending on the options used.

VCB Structure

```
typedef struct query_lu_0_to_3
{
   AP_UINT16       opcode;                 /* verb operation code          */
   unsigned char   reserv2;                /* reserved                     */
   unsigned char   format;                 /* reserved                     */
   AP_UINT16       primary_rc;             /* primary return code          */
   AP_UINT32       secondary_rc;           /* secondary return code        */
   unsigned char   *buf_ptr;               /* pointer to buffer            */
   AP_UINT32       buf_size;               /* buffer size                  */
   AP_UINT32       total_buf_size;         /* total buffer size required   */
   AP_UINT16       num_entries;            /* number of entries            */
   AP_UINT16       total_num_entries;      /* total number of entries      */
   unsigned char   list_options;           /* listing options              */
   unsigned char   reserv3;                /* reserved                     */
   unsigned char   pu_name[8];             /* PU name filter               */
   unsigned char   lu_name[8];             /* LU name                      */
   unsigned char   host_attachment;        /* host attachment filter       */
} QUERY_LU_0_TO_3;


typedef struct lu_0_to_3_summary
{
   AP_UINT16       overlay_size;           /* size of returned entry       */
   unsigned char   pu_name[8];             /* PU name                      */
   unsigned char   lu_name[8];             /* LU name                      */
   unsigned char   description[32];        /* resource description         */
   unsigned char   reserv1[16];            /* reserved                     */
   unsigned char   nau_address;            /* NAU address                  */
   unsigned char   lu_sscp_sess_active;    /* Is LU-SSCP session active    */
   unsigned char   appl_conn_active;       /* Is connection to appl active */
```

```
   unsigned char   plu_sess_active;          /* Is PLU-SLU session active    */
   unsigned char   host_attachment;          /* LU's host attachment         */
} LU_0_TO_3_SUMMARY;


typedef struct lu_0_to_3_detail
{
   AP_UINT16       overlay_size;             /* size of returned entry       */
   unsigned char   lu_name[8];               /* LU name                      */
   unsigned char   reserv1[2];               /* reserved                     */
   LU_0_TO_3_DET_DATA  det_data;             /* Determined data              */
   LU_0_TO_3_DEF_DATA  def_data;             /* Defined data                 */
} LU_0_TO_3_DETAIL;


typedef struct lu_0_to_3_det_data
{
   unsigned char   lu_sscp_sess_active;      /* Is LU-SSCP session active    */
   unsigned char   appl_conn_active;         /* Application is using LU      */
   unsigned char   plu_sess_active;          /* Is PLU-SLU session active    */
   unsigned char   host_attachment;          /* Host attachment              */
   SESSION_STATS   lu_sscp_stats;            /* reserved                     */
   SESSION_STATS   plu_stats;                /* reserved                     */
   unsigned char   plu_name[8];              /* PLU name                     */
   unsigned char   session_id[8];            /* Internal ID of PLU-SLU sess  */
   unsigned char   app_spec_det_data[256];   /* Application specified data   */
   unsigned char   app_type;                 /* Type of application using LU */
   unsigned char   sscp_id[6];               /* sscp id                      */
   unsigned char   reserva[19];              /* reserved                     */
} LU_0_TO_3_DET_DATA;


typedef struct session_stats
{
   AP_UINT16       rcv_ru_size;          /* session receive RU size          */
   AP_UINT16       send_ru_size;         /* session send RU size             */
   AP_UINT16       max_send_btu_size;    /* maximum send BTU size            */
   AP_UINT16       max_rcv_btu_size;     /* maximum rcv BTU size             */
   AP_UINT16       max_send_pac_win;     /* maximum send pacing window size  */
   AP_UINT16       cur_send_pac_win;     /* current send pacing window size  */
   AP_UINT16       max_rcv_pac_win;      /* maximum receive pacing window    */
                                         /*  size                            */
   AP_UINT16       cur_rcv_pac_win;      /* current receive pacing window    */
                                         /*  size                            */
   AP_UINT32       send_data_frames;     /* number of data frames sent       */
   AP_UINT32       send_fmd_data_frames; /* num fmd data frames sent         */
   AP_UINT32       send_data_bytes;      /* number of data bytes sent        */
```

```
   AP_UINT32       rcv_data_frames;      /* number of data frames received   */
   AP_UINT32       rcv_fmd_data_frames;  /* num fmd data frames received     */
   AP_UINT32       rcv_data_bytes;       /* number of data bytes received    */
   unsigned char   sidh;                 /* session ID high byte (from LFSID)*/
   unsigned char   sidl;                 /* session ID low byte (from LFSID) */
   unsigned char   odai;                 /* ODAI bit set                     */
   unsigned char   ls_name[8];           /* Link station name                */
   unsigned char   pacing_type;          /* type of pacing in use            */
} SESSION_STATS;


typedef struct lu_0_to_3_def_data
{
   unsigned char   description[32];          /* resource description         */
   unsigned char   reserv1[16];              /* reserved                     */
   unsigned char   nau_address;              /* LU NAU address               */
   unsigned char   pool_name[8];             /* LU Pool name                 */
   unsigned char   pu_name[8];               /* PU name                      */
   unsigned char   priority;                 /* LU priority                  */
   unsigned char   lu_model;                 /* LU model (type)              */
   unsigned char   sscp_id[6];               /* SSCP ID                      */
   AP_UINT16       timeout;                  /* Timeout                      */
   unsigned char   app_spec_def_data[16];    /* application-specified data   */
} LU_0_TO_3_DEF_DATA;
```

If the *app_type* parameter in the lu_0_to_3_det_data structure is set to AP_FMI_APPLICATION, the *app_spec_det_data* field contains the following structure:

```
typedef struct session_user_info
{
   unsigned char   user_name[32];            /* 3270 user name               */
   unsigned char   system_name[64];          /* computer name                */
   AP_UINT32       user_pid;                 /* process ID                   */
   AP_UINT32       user_type;                /* type of application using LU */
   AP_UINT32       user_uid;                 /* user ID                      */
   AP_UINT32       user_gid;                 /* group ID                     */
   unsigned char   user_gname[32];           /* group name                   */
   unsigned char   reserv4[32];              /* reserved                     */
} SESSION_USER_INFO;
```

If the *app_type* parameter in the lu_0_to_3_det_data structure is set to AP_RJE_WKSTN, the *app_spec_det_data* field contains the same structure as the 3270 structure above except that the *user_type* and *lu_model* parameters are set to AP_RJE_WKSTN and the *user_name*

parameter is replaced by a *workstation_name* parameter.

If the *app_type* parameter in the lu_0_to_3_det_data structure is set to AP_PU_CONCENTRATION, the *app_spec_det_data* field contains the same structure as the 3270 structure above except that the *app_type* parameter is set to AP_PU_CONCENTRATION and the *user_name* through *user_gname* parameters are replaced by a *pu_conc_downstream_lu* parameter.

If the *app_type* parameter in the lu_0_to_3_det_data structure is set to AP_LUA_APPLICATION, the *app_spec_det_data* field contains the same structure as the 3270 structure above except that the *app_type* parameter is set to AP_LUA_APPLICATION and the *user_name* through *user_gname* parameters are not returned.

If the *app_type* parameter in the lu_0_to_3_det_data structure is set to AP_TN_SERVER, the *app_spec_det_data* field contains the following structure:

```
typedef struct tn_server_session_user_info
{
   unsigned char   user_ip_address[16];      /* user's IP address            */
   AP_UINT16       port_number;              /* TCP/IP port number           */
   AP_UINT16       cb_number;                /* reserved                     */
   AP_UINT16       cfg_default;              /* using the default record?    */
   unsigned char   cfg_address[68];          /* address from config record   */
   AP_UINT16       cfg_format;               /* format of address            */
   unsigned char   tn3270_level;             /* TN3270 level used:           */
                                             /*   AP_LEVEL_TN3270            */
                                             /*   AP_LEVEL_TN3270E           */
   unsigned char   lu_select;                /* method of LU selection:      */
                                             /*   AP_GENERIC_LU              */
                                             /*   AP_SPECIFIC_LU             */
                                             /*   AP_ASSOCIATED_LU           */
   unsigned char   request_lu_name[8];       /* requested LU name or         */
                                             /* associated display LU name   */
                                             /*  (in EBCDIC)                 */
   unsigned char   reserv3[22];              /* reserved                     */
} TN_SERVER_SESSION_USER_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_LU_0_TO_3

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value `1`. To return as many entries as possible, specify `0`; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *lu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *lu_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

pu_name

PU name for which LU information is required. To list only information about LUs associated with a specific PU, specify the PU name. To obtain a complete list for all PUs, set this field to binary zeros.

lu_name

Name of the local LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

host_attachment

Host attachment filter. If the verb is issued to a running node, this parameter specifies whether to filter the returned information by whether the LUs are attached to the host directly or using DLUR. Possible values are:

AP_DIRECT_ATTACHED

Return information only on LUs directly attached to the host system.

AP_DLUR_ATTACHED

Return information only on LUs supported by DLUR.

AP_NONE

Return information about all LUs regardless of host attachment.

If the node is not running, this parameter is ignored; SNAplus2 returns information about all LUs regardless of host attachment.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

lu_0_to_3_summary.overlay_size

The size of the returned lu_0_to_3_summary structure, and therefore the offset to the start of the next entry in the data buffer.

lu_0_to_3_summary.pu_name

Name of the local PU used by the LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

lu_0_to_3_summary.lu_name

Name of the local LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with

EBCDIC spaces if the name is shorter than 8 characters.

lu_0_to_3_summary.description

A null-terminated text string describing the LU, as specified in the definition of the LU.

lu_0_to_3_summary.nau_address

Network accessible unit address of the LU. This is in the range 1-255.

lu_0_to_3_summary.lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

lu_0_to_3_summary.appl_conn_active

Specifies whether an application is using the LU. Possible values are:

AP_YES

An application is using the LU.

AP_NO

No application is using the LU.

lu_0_to_3_summary.plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

lu_0_to_3_summary.host_attachment

LU host attachment type.

When the verb is issued to a running node, this parameter takes one of the following values:

AP_DIRECT_ATTACHED

LU is directly attached to the host system.

AP_DLUR_ATTACHED

LU is supported by DLUR.

lu_0_to_3_detail.overlay_size

The size of the returned lu_0_to_3_detail structure, and therefore the offset to the start of the next entry in the data buffer.

lu_0_to_3_detail.lu_name

Name of the local LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

lu_0_to_3_detail.det_data.lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

lu_0_to_3_detail.det_data.appl_conn_active

Specifies whether an application is using the LU. Possible values are:

AP_YES

An application is using the LU.

AP_NO

No application is using the LU.

lu_0_to_3_detail.det_data.plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

lu_0_to_3_detail.det_data.host_attachment

LU host attachment type.

When the verb is issued to a running node, this parameter takes one of the following values:

AP_DIRECT_ATTACHED

LU is directly attached to the host system.

AP_DLUR_ATTACHED

LU is supported by DLUR.

For each of the two sessions (LU-SSCP session and PLU-SLU session), the session_stats structure contains the following parameters:

rcv_ru_size

Maximum receive RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

send_ru_size

Maximum send RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

max_send_btu_size

Maximum BTU size that can be sent.

max_rcv_btu_size

Maximum BTU size that can be received.

max_send_pac_win

Maximum size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_send_pac_win

Current size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

max_rcv_pac_win

Maximum size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_rcv_pac_win

Current size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

send_data_frames

Number of normal flow data frames sent.

send_fmd_data_frames

Number of normal flow FMD data frames sent.

send_data_bytes

Number of normal flow data bytes sent.

rcv_data_frames

Number of normal flow data frames received.

rcv_fmd_data_frames

Number of normal flow FMD data frames received.

rcv_data_bytes

Number of normal flow data bytes received.

sidh

Session ID high byte.

sidl

Session ID low byte.

odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station,

and sets it to one if the BIND sender is the node containing the secondary link station.

ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters.

pacing_type

Receive pacing type in use on the PLU-SLU session. Possible values are:

```
AP_NONE
AP_PACING_FIXED
```

lu_0_to_3_detail.det_data.plu_name

Name of the primary LU. This is an 8-byte type-A EBCDIC string, right-padded with spaces if the name is shorter than 8 characters. This name is reserved if the PLU-SLU session is inactive.

lu_0_to_3_detail.det_data.session_id

Eight byte internal identifier of the PLU-SLU session.

lu_0_to_3_detail.det_data.app_spec_det_data

The format of the data in this field depends on the value of the *app_type* field below, as follows:

- If *app_type* is set to AP_NONE or AP_LUA_APPLICATION, this field is reserved.
- If *app_type* is set to AP_PU_CONCENTRATION, the first 8 bytes of this field contain the LU name of the downstream LU currently using this local LU. This is an EBCDIC string, right-padded with spaces if the name is shorter than 8 characters. The remaining bytes are reserved.
- If *app_type* is set to AP_FMI_APPLICATION, this field is replaced by the session_user_info structure, containing information about the user of this LU. The structure consists of the following fields:

user_name

If the *user_type* parameter below is set to AP_RJE_WKSTN, this parameter specifies the name of the RJE workstation program that is using the LU. This is an ASCII string of 1-4 characters; the remaining characters are filled with spaces.

For other values of *user_type*, this parameter specifies the HP-UX user name with which the 3270 emulation program is running. This is an ASCII string of 1-32 characters.

system_name

The computer name on which the program is running.

user_pid

The process ID of the program using the LU.

user_type

The type of session (3270 display session, 3270 printer session, or RJE session) using the LU. Possible values are:

AP_3270_DISPLAY_MODEL_2

AP_3270_DISPLAY_MODEL_3

AP_3270_DISPLAY_MODEL_4

AP_3270_DISPLAY_MODEL_5

AP_PRINTER

AP_RJE_WKSTN

AP_SCS_PRINTER

AP_UNKNOWN

user_uid

The HP-UX user ID with which the program is running.

user_gid

The HP-UX group ID with which the program is running.

user_gname

The HP-UX group name with which the program is running. This is an ASCII string of 1-32 characters.

If *app_type* is set to AP_TN_SERVER, this field is replaced by the tn_server_session_user_info structure, containing information about the TN3270 program that is using this LU. The structure consists of the following fields:

user_ip_address

The dotted-decimal IP address of the computer where the TN3270 program is running.

port_number

The TCP/IP port number that the TN3270 program uses to access TN server.

cb_number

TN server control block number.

cfg_default

Specifies whether the TN3270 program is using an explicitly-defined TN server user record, or is using the configured default record. For more information about configuring a default TN server user record, see "DEFINE_TN3270_ACCESS". Possible values are:

AP_YES

The program is using the default record. The *cfg_address* and *cfg_format* parameters below are reserved.

AP_NO

The program is using an explicitly-defined record.

cfg_address

The TCP/IP address of the computer on which the TN3270 program runs, as defined in the configuration record that this user is using. This is a string of 1-64 characters, followed by a null character.

The address may be specified as a dotted-decimal IP address (such as 193.1.11.100), as a name (such as

newbox.this.co.uk), or as an alias (such as newbox); the format is indicated by the *cfg_format* parameter.

cfg_format

Specifies the format of the *cfg_address* parameter. Possible values are:

AP_ADDRESS_IP

IP address

AP_ADDRESS_FQN

Alias or fully qualified name

tn3270_level

Level of TN3270 support. Possible values are:

AP_LEVEL_TN3270

TN3270E protocols are disabled.

AP_LEVEL_TN3270E

TN3270E protocols are enabled.

lu_select

Method of LU selection. Possible values are:

AP_GENERIC_LU

The TN3270 program selected a generic display or printer LU.

AP_SPECIFIC_LU

The TN3270 program selected this LU specifically.

AP_ASSOCIATED_LU

This is a printer LU that has been associated with a display LU by a DEFINE_TN3270_ASSOCIATION verb, or a display LU that has been associated with a printer LU by a DEFINE_TN3270_ASSOCIATION verb. The LU is in use by the TN3270 through its association.

request_lu_name

Requested LU name or associated display LU name.

This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

lu_0_to_3_detail.det_data.app_type

The type of application, if any, that is using the LU. Possible values are:

AP_NONE

The LU is not in use.

AP_LUA_APPLICATION

The LU is being used by an LUA application.

AP_PU_CONCENTRATION

The LU is being used by a downstream LU using PU concentration.

AP_FMI_APPLICATION

The LU is being used by a 3270 emulation program or an RJE workstation; the *user_type* parameter in the session_user_info structure (described above) identifies the application type.

AP_TN_SERVER

The LU is being used by a TN3270 program accessing TN server.

lu_0_to_3_detail.det_data.sscp_id

A 6-byte field containing the SSCP ID received in the ACTPU for the PU used by this LU. If *lu_sscp_sess_active* is not AP_NO, this parameter will be all zeros.

lu_0_to_3_detail.def_data.description

A null-terminated text string describing the LU, as specified in the definition of the LU.

lu_0_to_3_detail.def_data.nau_address

Network accessible unit address of the LU, in the range 1-255.

lu_0_to_3_detail.def_data.pool_name

Name of the LU pool to which this LU belongs. This is an 8-byte EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. If the LU does not belong to a pool, this field is set to 8 binary zeros.

lu_0_to_3_detail.def_data.pu_name

Name of the PU (as specified on the DEFINE_LS verb) which this LU will use. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

lu_0_to_3_detail.def_data.priority

LU priority when sending to the host. This is set to one of the following:

```
AP_NETWORK
AP_HIGH
AP_MEDIUM
AP_LOW
```

lu_0_to_3_detail.def_data.lu_model

Type of the LU. This is set to one of the following:

AP_3270_DISPLAY_MODEL_2

AP_3270_DISPLAY_MODEL_3

AP_3270_DISPLAY_MODEL_4

AP_3270_DISPLAY_MODEL_5

AP_PRINTER

AP_SCS_PRINTER

AP_RJE_WKSTN

AP_UNKNOWN

lu_0_to_3_detail.def_data.sscp_id

Specifies the ID of the SSCP permitted to activate this LU. This is a 6-byte binary field. If this parameter is set to binary zeros, the LU may be activated by any SSCP.

lu_0_to_3_detail.def_data.timeout

Timeout for the LU, specified in seconds. If a timeout is

supplied and the user of the LU specified allow_timeout on the OPEN_LU_SSCP_SEC_RQ (or, in the case of PU concentration, on the downstream LU definition), then the LU will be deactivated after the PLU-SLU session is left inactive for this period and one of the following conditions applies:

- The session passes over a limited resource link.
- Another application wishes to use the LU before the session is used again.

If the timeout is set to zero, the LU will not be deactivated.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LU_LU_PASSWORD

QUERY_LU_LU_PASSWORD returns information about passwords used for session-level security verification between a local LU and a partner LU. It can be used to obtain information about the password for a specific partner LU or about passwords for multiple partner LUs, depending on the options used.

This verb returns password information in clear text. Be careful about how you use this verb, to ensure that unauthorized users are not given access to password information.

VCB Structure

```
typedef struct query_lu_lu_password
{
   AP_UINT16        opcode;                /* Verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* Primary return code          */
   AP_UINT32        secondary_rc;          /* Secondary return code        */
   unsigned char    *buf_ptr;              /* pointer to buffer            */
   AP_UINT32        buf_size;              /* buffer size                  */
   AP_UINT32        total_buf_size;        /* total buffer size required   */
   AP_UINT16        num_entries;           /* number of entries    *       */
   AP_UINT16        total_num_entries;     /* total number of entries      */
   unsigned char    list_options;          /* listing options              */
   unsigned char    reserv3;               /* reserved                     */
   unsigned char    lu_name[8];            /* LU name                      */
   unsigned char    lu_alias[8];           /* LU alias                     */
   unsigned char    plu_alias[8];          /* partner LU alias             */
   unsigned char    fqplu_name[17];        /* fully-qual. partner LU name  */
} QUERY_LU_LU_PASSWORD;


typedef struct password_info
{
   AP_UINT16        overlay_size;          /* size of returned entry       */
   unsigned char    plu_alias[8];          /* partner LU alias             */
   unsigned char    fqplu_name[17];        /* fully-qual. partner LU name  */
   unsigned char    description[32];       /* resource description         */
   unsigned char    reserv1[16];           /* reserved                     */
   unsigned char    password[8];           /* password                     */
   unsigned char    protocol_defined;      /* protocol defined             */
```

```
   unsigned char    protocol_in_use;         /* protocol in use              */
} PASSWORD_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_LU_LU_PASSWORD

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of partner LUs for which password information should be returned. To request a specific entry rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *plu_alias* or *fqplu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *plu_alias* or *fqplu_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

lu_name

LU name. This name is an 8-byte type-A EBCDIC character string. To indicate that the LU is identified by its LU alias instead of its LU name, set this parameter to 8 binary zeros, and specify the LU alias in the *lu_alias* parameter.

lu_alias

Locally defined LU alias. This is an 8-byte ASCII character string. This parameter is used only if *lu_name* is set to all zeros; it is ignored otherwise. To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to all zeros.

plu_alias

Partner LU alias. This is an 8-byte ASCII character string. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. To indicate that the partner LU is identified by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros, and specify the LU alias in the *fqplu_name* parameter.

fqplu_name

Fully qualified network name for the partner LU. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. This parameter is used only if *plu_alias* is set to all zeros; it is ignored otherwise.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

password_info.overlay_size

The size of the returned password_info structure, and therefore the offset to the start of the next entry in the data buffer.

password_info.plu_alias

Partner LU alias. This is an 8-byte ASCII character string.

password_info.fqplu_name

Fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

password_info.description

A null-terminated text string describing the LU-LU password, as specified in the definition of the password.

password_info.password

An encrypted version of the password supplied on a DEFINE_LU_LU_PASSWORD verb. This is an 8-byte hexadecimal string.

password_info.protocol_defined

Requested LU-LU verfication protocol defined for use with this partner LU. Possible values are:

AP_BASIC

Basic security protocols requested.

AP_ENHANCED

Enhanced security protocols requested.

AP_EITHER

Basic or enhanced security accepted.

password_info.protocol_in_use

LU-LU verfication protocol in use with this partner LU. Possible values are:

AP_BASIC

Basic security protocols in use.

AP_ENHANCED

Enhanced security protocols in use.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2

returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LU_ALIAS

The supplied *lu_alias* parameter did not match the alias of any configured LU.

AP_INVALID_LU_NAME

The supplied *lu_name* parameter did not match the name of any configured LU.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LU_POOL

QUERY_LU_POOL returns information about LU pools and the LUs that belong to them.

This verb can be used to obtain information about a specific LU or pool, or about multiple LUs or pools, depending on the options used.

VCB Structure

```
typedef struct query_lu_pool
{
   AP_UINT16        opcode;                 /* verb operation code        */
   unsigned char    reserv2;                /* reserved                   */
   unsigned char    format;                 /* reserved                   */
   AP_UINT16        primary_rc;             /* primary return code        */
   AP_UINT32        secondary_rc;           /* secondary return code      */
   unsigned char    *buf_ptr                /* pointer to buffer          */
   AP_UINT32        buf_size;               /* buffer size                */
   AP_UINT32        total_buf_size;         /* total buffer size required */
   AP_UINT16        num_entries;            /* number of entries          */
   AP_UINT16        total_num_entries;      /* total number of entries    */
   unsigned char    list_options;           /* listing options            */
   unsigned char    reserv3;                /* reserved                   */
   unsigned char    pool_name[8];           /* Pool name                  */
   unsigned char    lu_name[8];             /* LU name                    */
} QUERY_LU_POOL;

typedef struct lu_pool_summary
{
   AP_UINT16        overlay_size;           /* size of returned entry     */
   unsigned char    pool_name[8];           /* Pool name                  */
   unsigned char    description[32];        /* resource description       */
   unsigned char    reserv1[16];            /* reserved                   */
   AP_UINT16        num_active_lus;         /* number of active lus       */
} LU_POOL_SUMMARY;

typedef struct lu_pool_detail
{
   AP_UINT16        overlay_size;           /* size of returned entry     */
   unsigned char    pool_name[8];           /* Pool name                  */
   unsigned char    description[32];        /* resource description       */
```

```
   unsigned char    reserv1[16];              /* reserved                   */
   unsigned char    lu_name[8];               /* LU name                    */
   unsigned char    lu_sscp_sess_active;      /* Is LU-SSCP session active  */
   unsigned char    appl_conn_active;         /* Is appl connection open    */
   unsigned char    plu_sess_active;          /* Is PLU-SLU session active  */
} LU_POOL_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_LU_POOL

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof() function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of entries for which data should be returned. If *list_options* is set to AP_SUMMARY, each entry is a single LU pool; if *list_options* is set to AP_DETAIL, each entry is an LU in a pool (or an entry indicating an empty LU pool).

To request a specific entry rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should

begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only (list LU pools).

AP_DETAIL

Detailed information (list individual LUs in LU pools).

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *pool_name* and *lu_name* parameters.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *pool_name* and *lu_name* parameters.

The list is ordered by *pool_name* and then by *lu_name*. For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

pool_name

Name of LU pool. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. This is an 8-byte EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

lu_name

LU name. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST or AP_SUMMARY. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

To obtain information about all LUs in a pool, set *pool_name* to the name of the pool, set *num_entries* to

0, and set *lu_name* to 8 binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

lu_pool_summary.overlay_size

The size of the returned lu_pool_data structure, and therefore the offset to the start of the next entry in the data buffer.

lu_pool_summary.pool_name

Name of LU pool. This is an 8-byte EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

lu_pool_summary.description

A null-terminated text string describing the LU pool,

as specified in the definition of the pool.

lu_pool_summary.num_active_lus

Number of LUs in the pool that are active.

lu_pool_detail.overlay_size

The size of the returned lu_pool_data structure, and therefore the offset to the start of the next entry in the data buffer.

lu_pool_detail.pool_name

Name of LU pool to which the LU belongs. This is an 8-byte EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

lu_pool_detail.description

A null-terminated text string describing the LU pool, as specified in the definition of the pool.

lu_pool_detail.lu_name

LU name of the LU. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. If a single lu_pool_detail structure is returned for a particular pool name with a string of 8 binary zeros for the LU name, this indicates that the specified pool is empty.

lu_pool_detail.lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

lu_pool_detail.appl_conn_active

Specifies whether an application is using the LU. Possible values are:

AP_YES

An application is using the LU.

AP_NO

No application is using the LU.

lu_pool_detail.plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is inactive.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_POOL_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *pool_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_LU62_TIMEOUT

The QUERY_LU62_TIMEOUT verb returns information about the definition of an LU type 6.2 session timeout that was defined previously with a DEFINE_LU62_TIMEOUT verb.

The information is returned as a list. To obtain information about a specific timeout, or about several timeout values, specify values for the *resource_type* and *resource_name* parameters. If the *list_options* parameter is set to AP_FIRST_IN_LIST, the *resource_type* and *resource_name* parameters are ignored. The returned list is ordered on *resource_type* and then on *resource_name*.

For *resource_type*, the ordering is:

1. Global timeouts
2. Local LU timeouts
3. Partner LU timeouts
4. Mode timeouts

For *resource_name*, the ordering is by:

1. Name length
2. By ASCII lexicographical ordering for names of the same length

If the *list_options* parameter is set to AP_LIST_FROM_NEXT, the returned list starts for the next entry according to the defined ordering (whether or not the specified entry exists).

VCB Structure

```
typedef struct query_lu62_timeout
{
  AP_UINT16      opcode;                  /* verb operation code       */
  unsigned char  reserv2;
  unsigned char  format;                  /* reserved                  */
  AP_UINT16      primary_rc;              /* primary return code       */
  AP_UINT32      secondary_rc;            /* secondary return code     */
  unsigned char *buf_ptr;                 /* buffer pointer            */
  AP_UINT32      buf_size;                /* buffer size               */
  AP_UINT32      total_buf_size;          /* total buffer size         */
  AP_UINT16      num_entries;             /* number of entries         */
```

```
  AP_UINT16      total_num_entries;          /* total number of entries    */
  unsigned char  list_options;               /* list options               */
  unsigned char  reserv3;                    /* reserved                   */
  unsigned char  resource_type;              /* resource type              */
  unsigned char  resource_name[17];          /* resource name              */
} QUERY_LU62_TIMEOUT;

typedef struct lu62_timeout_data
{
  AP_UINT16      overlay_size;               /* overlay size               */
  unsigned char  resource_type;              /* resource type              */
  unsigned char  resource_name[17];          /* resource name              */
  AP_UINT16      timeout;                    /* timeout                    */
} LU62_TIMEOUT_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_LU62_TIMEOUT

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of entries for which data should be returned. To request data for a specific entry rather than a range, specify the value `1`. To return as many entries as possible, specify `0`; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of entries from which SNAplus2 begins to return data. The list is ordered by *resource_type* in the order AP_GLOBAL_TIMEOUT, AP_LOCAL_LU_TIMEOUT, AP_PARTNER_LU_TIMEOUT, AP_MODE_TIMEOUT, then by *resource_name* in order of the name length, then by ASCII lexicographical ordering for names of the same length.

Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of the *resource_type* and *resource_name* parameters

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of the *resource_type* and *resource_name* parameters

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

resource_type

Specifies the type of timeout being queried. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Possible values are:

AP_GLOBAL_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local node.

AP_LOCAL_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local LU specified in the *resource_name* parameter.

AP_PARTNER_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions to the partner LU specified in the *resource_name* parameter.

AP_MODE_TIMEOUT

Timeout applies to all LU 6.2 sessions using the mode specified in the *resource_name* parameter.

resource_name

Name of the resource being queried. This value can be one of the following:

- If *resource_type* is set to AP_GLOBAL_TIMEOUT, do not specify this parameter.
- If *resource_type* is set to AP_LOCAL_LU_TIMEOUT, only the first 8 bytes of *resource_name* are valid and should be set to the name of the local LU. This is an 8-byte alphanumeric type-A EBCDIC string starting with a letter, padded to the right with EBCDIC spaces. Set the remaining nine bytes to all zeros.
- If *resource_type* is set to AP_PARTNER_LU_TIMEOUT, all 17 bytes of resource_name are valid and should be set to the fully-qualified name of the partner LU which is padded on the right with EBCDIC spaces. The name consists of a 1-8 type-A character network name, followed by an EBCDIC dot (period) character, followed by a 1-8 type-A character partner LU name.
- If *resource_type* is set to AP_MODE_TIMEOUT, only the first 8 bytes of resource_name are valid and should be set to the name of the mode. This is an 8-byte alphanumeric type-A EBCDIC string starting with a letter, padded to the right with EBCDIC spaces. Set he remaining 9 bytes to all zeros.

This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

resource_type

The type of the timeout. Possible values are:

AP_GLOBAL_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local node. The *resource_name* parameter is set to all zeros.

AP_LOCAL_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions for the local LU indicated by the *resource_name* parameter.

AP_PARTNER_LU_TIMEOUT

Timeout applies to all LU 6.2 sessions to the partner LU indicated by the *resource_name* parameter.

AP_MODE_TIMEOUT

Timeout applies to all LU 6.2 sessions using the mode indicated by the *resource_name* parameter.

resource_name

Name of the resource. This name is a local LU, a partner LU, or a mode, depending on the value of the *resource_type* parameter. This parameter is set to zeros if *resource_type* is set to AP_GLOBAL_TIMEOUT.

timeout

Timeout period in seconds. A value of 0 (zero) indicates that the session times out immediately after it becomes free.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_RESOURCE_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name and type, but the combination of *resource_type* and *resource_name* did not match any that are configured.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_MDS_APPLICATION

QUERY_MDS_APPLICATION returns a list of applications that have registered for MDS-level messages by issuing the MS verb REGISTER_MS_APPLICATION. For more information about this verb, see the *HP-UX SNAplus2 MS Programmers Guide*.

This verb can be used to obtain information about a specific application or about multiple applications, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_mds_application
{
   AP_UINT16        opcode;               /* verb operation code           */
   unsigned char    reserv2;              /* reserved                      */
   unsigned char    format;               /* reserved                      */
   AP_UINT16        primary_rc;           /* primary return code           */
   AP_UINT32        secondary_rc;         /* secondary return code         */
   unsigned char    *buf_ptr;             /* pointer to buffer             */
   AP_UINT32        buf_size;             /* buffer size                   */
   AP_UINT32        total_buf_size;       /* total buffer size required    */
   AP_UINT16        num_entries;          /* number of entries             */
   AP_UINT16        total_num_entries;    /* total number of entries       */
   unsigned char    list_options;         /* listing options               */
   unsigned char    reserv3;              /* reserved                      */
   unsigned char    application[8];       /* application                   */
} QUERY_MDS_APPLICATION;


typedef struct mds_application_data
{
   AP_UINT16        overlay_size;         /* size of returned entry        */
   unsigned char    application[8];       /* application name              */
   AP_UINT16        max_rcv_size;         /* max data size appl can receive */
   unsigned char    reserva[20];          /* reserved                      */
}MDS_APPLICATION_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_MDS_APPLICATION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of applications for which data should be returned. To request data for a specific application rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of applications from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the application parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the application parameter.

For more information about how the list is ordered and

how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

application

Application name for which information is required, or the name to be used as an index into the list. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

mds_application_data.overlay_size

The size of the returned mds_application_data structure, and therefore the offset to the start of the

next entry in the data buffer.

mds_application_data.application

Name of registered application. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

mds_application_data.max_rcv_size

The maximum number of bytes that the application can receive in one message (this is specified when an application registers with MDS). For more information about MDS-level application registration, refer to the *HP-UX SNAplus2 MS Programmers Guide*.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_APPLICATION_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *application* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_MDS_STATISTICS

QUERY_MDS_STATISTICS returns Management Services statistics. This verb can be used to gauge the level of MDS routing traffic. The information can also be used to determine the required size of the send alert queue, which is configured using the DEFINE_NODE verb.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_mds_statistics
{
   AP_UINT16       opcode;                   /* verb operation code         */
   unsigned char   reserv2;                  /* reserved                    */
   unsigned char   format;                   /* reserved                    */
   AP_UINT16       primary_rc;               /* primary return code         */
   AP_UINT32       secondary_rc;             /* secondary return code       */
   AP_UINT32       alerts_sent;              /* number of alert sends       */
   AP_UINT32       alert_errors_rcvd;        /* error messages received     */
                                             /*  for alert sends            */
   AP_UINT32       uncorrelated_alert_errors; /* uncorrelated alert errors  */
                                             /*  received                   */
   AP_UINT32       mds_mus_rcvd_local;       /* number of MDS_MUs received  */
                                             /*  from local applications    */
   AP_UINT32       mds_mus_rcvd_remote;      /* number of MDS_MUs received  */
                                             /*  from remote applications   */
   AP_UINT32       mds_mus_delivered_local;  /* number of MDS_MUs delivered */
                                             /*  to local applications      */
   AP_UINT32       mds_mus_delivered_remote; /* number of MDS_MUs delivered */
                                             /*  to remote applications     */
   AP_UINT32       parse_errors;             /* number of MDS_MUs received  */
                                             /*  with parse errors          */
   AP_UINT32       failed_deliveries;        /* number of MDS_MUs where     */
                                             /*  delivery failed            */
   AP_UINT32       ds_searches_performed;    /* number of DS searches       */
                                             /*  performed                  */
   AP_UINT32       unverified_errors;        /* number of unverified errors */
   unsigned char   reserva[20];              /* reserved                    */
} QUERY_MDS_STATISTICS;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_MDS_STATISTICS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

alerts_sent

Number of locally originated alerts sent using the MDS transport system.

alert_errors_rcvd

Number of error messages received by MDS indicating a delivery failure for a message containing an alert.

uncorrelated_errors_rcvd

Number of error messages received by MDS indicating a delivery failure for a message containing an alert. Delivery failure occurs when the error message could not be correlated to an alert on the MDS send alert queue. MDS maintains a fixed-size queue where it caches alerts sent to the problem determination focal point. Once the queue reaches maximum size, the oldest alert will be discarded and replaced by the new alert. If a delivery error message is received, MDS attempts to correlate the error message to a cached alert so that the alert may be held until the problem determination focal point is restored.

NOTE The two counts *alert_errors_rcvd* and *uncorrelated_errors_rcvd* can be used to check that the size of the send alert queue (specified on DEFINE_NODE) is appropriate. If the value of

uncorrelated_errors_rcvd increases over time, this indicates that the send alert queue size is too small.

mds_mus_rcvd_local

Number of MDS_MUs received from local applications.

mds_mus_rcvd_remote

Number of MDS_MUs received from remote nodes using the MDS_RECEIVE and MSU_HANDLER transaction programs.

mds_mus_delivered_local

Number of MDS_MUs successfully delivered to local applications.

mds_mus_delivered_remote

Number of MDS_MUs successfully delivered to a remote node using the MDS_SEND transaction program.

parse_errors

Number of MDS_MUs received which contained header format errors.

failed_deliveries

Number of MDS_MUs this node failed to deliver.

ds_searches_performed

Number of Directory Services searches used to locate the next hop for an MDS_MU. (Significant for network nodes only).

unverified_errors

Number of routing errors due to using unverified (local Directory Services search) data for determining the next hop for an MDS_MU. Each time one of these errors occurs, Directory Services must repeat the search using either a Central Directory Search or a broadcast search mechanism. (Significant for network nodes only).

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node configuration does not support it, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support MS network management functions; this is defined by the *mds_supported* parameter on the DEFINE_NODE verb.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_MODE

QUERY_MODE returns information about modes that a local LU is using, or has used, with partner LUs.

This verb can be used to obtain information about a specific partner LU-mode combination or about multiple modes, and about modes for which sessions are currently active or about all modes that have been used, depending on the options used. This verb returns information about current usage of the modes and LUs, not about their definition; use QUERY_MODE_DEFINITION to obtain the definition of the modes and LUs.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_mode
{
   AP_UINT16       opcode;              /* verb operation code             */
   unsigned char   reserv2;             /* reserved                        */
   unsigned char   format;              /* reserved                        */
   AP_UINT16       primary_rc;          /* primary return code             */
   AP_UINT32       secondary_rc;        /* secondary return code           */
   unsigned char   *buf_ptr;            /* pointer to buffer               */
   AP_UINT32       buf_size;            /* buffer size                     */
   AP_UINT32       total_buf_size;      /* total buffer size required      */
   AP_UINT16       num_entries;         /* number of entries               */
   AP_UINT16       total_num_entries;   /* total number of entries         */
   unsigned char   list_options;        /* listing options                 */
   unsigned char   reserv3;             /* reserved                        */
   unsigned char   lu_name[8];          /* LU name                         */
   unsigned char   lu_alias[8];         /* LU alias                        */
   unsigned char   plu_alias[8];        /* partner LU alias                */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name */
   unsigned char   mode_name[8];        /* mode name                       */
   unsigned char   active_sessions;     /* active sessions only filter     */
} QUERY_MODE;

typedef struct mode_summary
{
   AP_UINT16       overlay_size;        /* size of returned entry          */
   unsigned char   mode_name[8];        /* mode name                       */
```

```
   unsigned char   description[32];       /* resource description            */
   unsigned char   reserv2[16];           /* reserved                        */
   AP_UINT16       sess_limit;            /* current session limit           */
   AP_UINT16       act_sess_count;        /* currently active sessions count */
   unsigned char   fqplu_name[17];        /* fully-qualified partner LU name */
   unsigned char   reserv1[3];            /* reserved                        */
} MODE_SUMMARY;

typedef struct mode_detail
{
   AP_UINT16       overlay_size;          /* size of returned entry          */
   unsigned char   mode_name[8];          /* mode name                       */
   unsigned char   description[32];       /* resource description            */
   unsigned char   reserv2[16];           /* reserved                        */
   AP_UINT16       sess_limit;            /* session limit                   */
   AP_UINT16       act_sess_count;        /* currently active sessions count */
   unsigned char   fqplu_name[17];        /* fully-qualified partner LU name */
   unsigned char   reserv1[3];            /* reserved                        */
   AP_UINT16       min_conwinners_source; /* minimum conwinner sess limit    */
   AP_UINT16       min_conwinners_target; /* minimum conloser sess limit     */
   unsigned char   drain_source;          /* drain source?                   */
   unsigned char   drain_partner;         /* drain partner?                  */
   AP_UINT16       auto_act;              /* auto activated conwinner        */
                                          /*  session limit                  */
   AP_UINT16       act_cw_count;          /* active conwinner sessions count */
   AP_UINT16       act_cl_count;          /* active conloser sessions count  */
   unsigned char   sync_level;            /* synchronization level           */
   unsigned char   default_ru_size;       /* default RU size to maximize     */
                                          /*  performance                    */
   AP_UINT16       max_neg_sess_limit;    /* maximum negotiated session limit*/
   AP_UINT16       max_rcv_ru_size;       /* maximum receive RU size         */
   AP_UINT16       pending_session_count; /* pending sess count for mode     */
   AP_UINT16       termination_count;     /* termination count for mode      */
   AP_UINT16       implicit;              /* implicit or explicit entry      */
   unsigned char   reserva[15];           /* reserved                        */
} MODE_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_MODE

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of modes for which data should be returned. To request data for a specific mode rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

`AP_SUMMARY`

Summary information only.

`AP_DETAIL`

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

`AP_FIRST_IN_LIST`

Start at the first entry in the list (the first partner LU and mode for the specified local LU).

`AP_LIST_INCLUSIVE`

Start at the entry specified by the supplied partner LU

name and mode name.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the supplied partner LU name and mode name.

For AP_FIRST_IN_LIST, the entry used as the index into the list is defined by the combination of *lu_name* (or *lu_alias*) and *fqplu_name* (or *plu_alias*). If *fqplu_name* or *plu_alias* is not specified, the entry used as the index is *lu_name* (or *lu_alias*).

For AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT, the entry used as the index into the list is defined by the combination of *lu_name* (or *lu_alias*), *fqplu_name* (or *plu_alias*) and *mode_name* specified. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

lu_name

LU name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To specify that the LU is identified by its alias rather than its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter.

lu_alias

Locally defined LU alias. This parameter is used only if *lu_name* is set to 8 binary zeros; it is ignored otherwise.

The alias is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the

fully qualified LU name for the partner LU. To specify that the LU is identified by its LU name rather than its alias, set this parameter to 8 binary zeros and specify the LU name in the following parameter.

fqplu_name

Fully qualified network name for the partner LU. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. This parameter is used only if *plu_alias* is set to 8 binary zeros; it is ignored otherwise.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Mode name which designates the network properties for a group of sessions. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

active_sessions

Specifies whether to return information only on modes for which sessions are active, or on all modes. Possible values are:

AP_YES

Return information only on modes for which sessions are currently active.

AP_NO

Return information about all modes for which sessions are active or have been active.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

mode_summary.overlay_size

The size of the returned mode_summary structure, and therefore the offset to the start of the next entry in the data buffer.

mode_summary.mode_name

Mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

mode_summary.description

A null-terminated text string describing the mode, as specified in the definition of the mode.

mode_summary.sess_limit

Current session limit.

mode_summary.act_sess_count

Total number of active sessions between the specified local LU and partner LU using the mode.

mode_summary.fqplu_name

Fully qualified name of the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_detail.overlay_size

The size of the returned `mode_detail` structure, and therefore the offset to the start of the next entry in the data buffer.

mode_detail.mode_name

Mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

mode_detail.description

A null-terminated text string describing the mode, as specified in the definition of the mode.

mode_detail.sess_limit

Current session limit.

mode_detail.act_sess_count

Total number of active sessions between the specified local LU and partner LU using the mode.

mode_detail.fqplu_name

Fully qualified name of the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_detail.min_conwinners_source

Specifies the minimum number of sessions on which the local LU is the contention winner.

mode_detail.min_conwinners_target

Specifies the minimum number of sessions on which the local LU is the contention loser.

mode_detail.drain_source

Specifies whether the local LU satisfies waiting session requests before deactivating a session when session limits are changed or reset. Possible values are:

AP_YES

Waiting session requests will be satisfied before sessions are deactivated.

AP_NO

Waiting session requests will not be satisfied.

mode_detail.drain_partner

Specifies whether the partner LU satisfies waiting session requests before deactivating a session when session limits are changed or reset. Possible values are:

AP_YES

Waiting session requests will be satisfied before sessions are deactivated.

AP_NO

Waiting session requests will not be satisfied.

mode_detail.auto_act

Number of contention winner sessions that are automatically activated following the CNOS exchange with the partner LU.

mode_detail.act_cw_count

Number of active contention winner sessions using this mode. (The local LU does not need to “bid” before using one of these sessions.)

mode_detail.act_cl_count

Number of active, contention loser sessions using this mode. (The local LU must “bid” before using one of these sessions.)

mode_detail.sync_level

Specifies the synchronization level supported by the mode. Possible values are:

AP_CONFIRM

The mode supports synchronization using the CONFIRM and CONFIRMED verbs.

AP_SYNCPT

The mode supports Syncpoint functions.

AP_NONE

The mode does not support synchronization.

mode_detail.default_ru_size

Specifies whether the default upper bound for the maximum RU size will be used. Possible values are:

AP_YES

SNAplus2 ignores the maximum RU size specified in the definition of the mode, and sets the upper bound for the maximum RU size to the largest value that can be accommodated in the link BTU size.

AP_NO

SNAplus2 uses the maximum RU size specified in the definition of the mode.

mode_detail.max_neg_sess_limit

Maximum negotiable session limit. Specifies the maximum session limit that a local LU can use with this mode name during its CNOS processing as the target LU.

mode_detail.max_rcv_ru_size

Maximum received RU size.

mode_detail.pending_session_count

Specifies the number of sessions pending (waiting for session activation to complete).

mode_detail.termination_count

If a previous CNOS verb has set the mode session limit to zero, but sessions are still active because conversations were using them or waiting to use them, this parameter specifies the number of sessions that have not yet been deactivated.

mode_detail.implicit

Specifies whether the entry was created by an implicit or explicit definition. Possible values are:

AP_YES

The entry is an implicit entry.

AP_NO

The entry is an explicit entry.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

AP_INVALID_LU_ALIAS

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_alias* parameter was not valid.

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_MODE_NAME

The *list_options* parameter was set to

AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *mode_name* parameter was not valid.

AP_INVALID_PLU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but one of the following conditions applies:

- The *fqplu_name* parameter does not match the name of any of this local LU's partners.
- No sessions have been active (since the node was last started) for the specified combination of local LU, partner LU, and mode.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_MODE_DEFINITION

QUERY_MODE_DEFINITIONreturns information about modes defined using DEFINE_MODE, or about SNA-defined modes.

This verb can be used to obtain either summary or detailed information, about a specific mode or about multiple modes, depending on the options used. It returns information about the definition of the modes, not about their current usage; use QUERY_MODE to obtain information about the current usage of a mode by local and partner LUs.

This verb cannot be used to return information about the default COS name that will be used for any unrecognized mode names; use QUERY_MODE_TO_COS_MAPPING to do this.

VCB Structure

```
typedef struct query_mode_definition
{
   AP_UINT16         opcode;                 /* verb operation code        */
   unsigned char     reserv2;                /* reserved                   */
   unsigned char     format;                 /* reserved                   */
   AP_UINT16         primary_rc;             /* primary return code        */
   AP_UINT32         secondary_rc;           /* secondary return code      */
   unsigned char     *buf_ptr;               /* pointer to buffer          */
   AP_UINT32         buf_size;               /* buffer size                */
   AP_UINT32         total_buf_size;         /* total buffer size required */
   AP_UINT16         num_entries;            /* number of entries          */
   AP_UINT16         total_num_entries;      /* total number of entries    */
   unsigned char     list_options;           /* listing options            */
   unsigned char     reserv3;                /* reserved                   */
   unsigned char     mode_name[8];           /* mode name                  */
} QUERY_MODE_DEFINITION;


typedef struct mode_def_summary
{
   AP_UINT16         overlay_size;           /* size of returned entry     */
   unsigned char     mode_name[8];           /* mode name                  */
   unsigned char     description[32];         /* resource description      */
   unsigned char     reserv1[16];            /* reserved                   */
} MODE_DEF_SUMMARY;
```

```
typedef struct mode_def_detail
{
   AP_UINT16         overlay_size;            /* size of returned entry     */
   unsigned char     mode_name[8];            /* mode name                  */
   MODE_CHARS        mode_chars;              /* mode characteristics       */
} MODE_DEF_DETAIL;

typedef struct mode_chars
{
   unsigned char     description[32];          /* resource description       */
   unsigned char     reserv2[16];             /* reserved                   */
   AP_UINT16         max_ru_size_upp;         /* maximum RU size upper bound*/
   unsigned char     receive_pacing_win;      /* receive pacing window      */
   unsigned char     default_ru_size;         /* default RU size to         */
                                              /*  maximize performance      */
   AP_UINT16         max_neg_sess_lim;        /* maximum negotiable session */
                                              /*  limit                     */
   AP_UINT16         plu_mode_session_limit;  /* LU-mode session limit      */
   AP_UINT16         min_conwin_src;          /* minimum source contention  */
                                              /*  winner sessions           */
   unsigned char     cos_name[8];             /* class of service name      */
   unsigned char     cryptography;            /* cryptography (reserved)    */
   unsigned char     compression;             /* Compression (reserved)     */
   AP_UINT16         auto_act;                /* number of sessions to be   */
                                              /* activated automatically    */
   AP_UINT16         min_conloser_src;        /* minimum source contention  */
                                              /*  loser                     */
   AP_UINT16         max_ru_size_low;         /* maximum RU size lower bound*/
   AP_UINT16         max_receive_pacing_win;  /* maximum receive pacing     */
                                              /*  window                    */
} MODE_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_MODE_DEFINITION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and

should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of modes for which data should be returned. To request data for a specific mode rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

`AP_SUMMARY`

Summary information only.

`AP_DETAIL`

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

`AP_FIRST_IN_LIST`

Start at the first entry in the list.

`AP_LIST_INCLUSIVE`

Start at the entry specified by the *mode_name* parameter.

`AP_LIST_FROM_NEXT`

Start at the entry immediately following the entry specified by the *mode_name* parameter.

For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”. This verb differs from other QUERY_* verbs in that the modes are listed in the order they are created.

mode_name

Mode name which designates the network properties for a group of sessions. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

mode_def_summary.overlay_size

The size of the returned `mode_def_summary` structure, and therefore the offset to the start of the next entry in the data buffer.

`mode_def_summary.mode_name`

Mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

`mode_def_summary.description`

A null-terminated text string describing the mode, as specified in the definition of the mode.

`mode_def_detail.overlay_size`

The size of the returned `mode_def_detail` structure, and therefore the offset to the start of the next entry in the data buffer.

`mode_def_detail.mode_name`

Mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

`mode_def_detail.mode_chars.description`

A null-terminated text string describing the mode, as specified in the definition of the mode.

`mode_def_detail.mode_chars.max_ru_size_upp`

Upper boundary for the maximum RU size to be used on sessions with this mode name. The value is used when the maximum RU size is negotiated during session activation.

Range: `256-61,440`. This field is ignored if the *`default_ru_size`* parameter (see below) is set to AP_YES.

`mode_def_detail.mode_chars.receive_pacing_win`

Session pacing window for sessions using this mode. For fixed pacing, this is the maximum number of frames that can be received from the partner LU before the local LU must send a response; for adaptive pacing, this value is used as an initial receive window size.

SNAplus2 always uses adaptive pacing unless the adjacent node specifies that it is not supported.

Range is 1-63, or zero to specify no pacing window (that is, an unlimited number of frames can be received, and no response is required).

mode_def_detail.mode_chars.default_ru_size

Specifies whether a default upper bound for the maximum RU size will be used. Possible values are:

AP_YES

SNAplus2 ignores the *max_ru_size_upp* parameter, and sets the upper bound for the maximum RU size to the largest value that can be accommodated in the link BTU size.

AP_NO

SNAplus2 uses the *max_ru_size_upp* parameter to define the maximum RU size.

mode_def_detail.mode_chars.max_neg_sess_lim

Maximum number of sessions allowed on this mode between any local LU and partner LU. Range: 1-32,767, or zero to specify no implicit CNOS exchange.

mode_def_detail.mode_chars.plu_mode_session_limit

Default session limit for this mode. This limits the number of sessions on this mode between any one local LU and partner LU pair. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. Range: 1-32,767, or zero to specify no implicit CNOS exchange.

mode_def_detail.mode_chars.min_conwin_src

Minimum number of contention winner sessions that a local LU using this mode can activate. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. Range: 1-32,767, or zero to specify no implicit CNOS exchange.

mode_def_detail.mode_chars.cos_name

Name of the class of service to request when activating sessions on this mode. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

mode_def_detail.mode_chars.auto_act

Specifies how many sessions will be activated automatically for this mode. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. This value is in the range 0-32,767.

mode_def_detail.mode_chars.min_conloser_src

Minimum number of contention loser sessions that can be activated by any one local LU that uses this mode. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. This value is in the range 0-32,767.

mode_def_detail.mode_chars.max_ru_size_low

Lower bound for the maximum size of RUs sent and received on sessions that use this mode.

This value is in the range 256-61,440 or zero, which means that there is no lower bound.

mode_def_detail.mode_chars.max_receive_pacing_win

Maximum session pacing window for sessions in this mode. For adaptive pacing, this value is used to limit the receive pacing window that the session will grant. For fixed pacing, this parameter is not used. (SNAplus2 always uses adaptive pacing unless the adjacent node specifies that it does not support it.)

This value is in the range 0-32,767 or zero, which means there is no limit for the pacing window.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_MODE_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *mode_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_MODE_TO_COS_MAPPING

QUERY_MODE_TO_COS_MAPPING returns information about the COS (class of service) associated with a particular mode. This verb can be used to obtain information about a specific mode or about multiple modes, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_mode_to_cos_mapping
{
   AP_UINT16        opcode;                 /* verb operation code         */
   unsigned char    reserv2;                /* reserved                    */
   unsigned char    format;                 /* reserved                    */
   AP_UINT16        primary_rc;             /* primary return code         */
   AP_UINT32        secondary_rc;           /* secondary return code       */
   unsigned char    *buf_ptr;               /* pointer to buffer           */
   AP_UINT32        buf_size;               /* buffer size                 */
   AP_UINT32        total_buf_size;         /* total buffer size required  */
   AP_UINT16        num_entries;            /* number of entries           */
   AP_UINT16        total_num_entries;      /* total number of entries     */
   unsigned char    list_options;           /* listing options             */
   unsigned char    reserv3;                /* reserved                    */
   unsigned char    mode_name[8];           /* mode name                   */
} QUERY_MODE_TO_COS_MAPPING;


typedef struct mode_to_cos_mapping_data
{
   AP_UINT16        overlay_size;           /* size of returned entry      */
   unsigned char    mode_name[8];           /* mode name                   */
   unsigned char    cos_name[8];            /* cos name                    */
   unsigned char    reserva[20];            /* reserved                    */
} MODE_TO_COS_MAPPING_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_MODE_TO_COS_MAPPING

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of modes for which data should be returned. To request data for a specific mode rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of modes from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *mode_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *mode_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

mode_name

Mode name for which information is required, or the name to be used as an index into the list. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The mode name is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To return information about the default COS that is used for any unrecognized mode names, set this parameter to 8 binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

mode_to_cos_mapping_data.overlay_size

The size of the returned mode_to_cos_mapping_data structure, and therefore the offset to the start of the next entry in the data buffer.

mode_to_cos_mapping_data.mode_name

Mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

mode_to_cos_mapping_data.cos_name

Class of service name associated with the mode name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_MODE_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *mode_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_NMVT_APPLICATION

QUERY_NMVT_APPLICATION returns a list of applications that have registered for NMVT-level messages by issuing the MS verb REGISTER_NMVT_APPLICATION. For more information about this verb, see the *HP-UX SNAplus2 MS Programmers Guide*.

This verb can be used to obtain information about a specific application or about multiple applications, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_nmvt_application
{
   AP_UINT16         opcode;                 /* Verb operation code       */
   unsigned char     reserv2;                /* reserved                  */
   unsigned char     format;                 /* reserved                  */
   AP_UINT16         primary_rc;             /* primary return code       */
   AP_UINT32         secondary_rc;           /* secondary return code     */
   unsigned char     *buf_ptr;               /* pointer to buffer         */
   AP_UINT32         buf_size;               /* buffer size               */
   AP_UINT32         total_buf_size;         /* total buffer size required*/
   AP_UINT16         num_entries;            /* number of entries         */
   AP_UINT16         total_num_entries;      /* total number of entries   */
   unsigned char     list_options;           /* listing options           */
   unsigned char     reserv3;                /* reserved                  */
   unsigned char     application[8];         /* application               */
} QUERY_NMVT_APPLICATION;


typedef struct nmvt_application_data
{
   AP_UINT16         overlay_size;           /* size of returned entry    */
   unsigned char     application[8];         /* application name          */
   AP_UINT16         ms_vector_key_type;     /* MS vector key accepted    */
                                             /*  by appl                  */
   unsigned char     conversion_required;    /* is conversion to MDS_MU   */
                                             /*  required                 */
   unsigned char     reserv[5];              /* reserved                  */
   unsigned char     reserva[20];            /* reserved                  */
}NMVT_APPLICATION_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_NMVT_APPLICATION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of applications for which data should be returned. To request data for a specific application rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of applications from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the application parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the application parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

application

Application name. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST. The name is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

nmvt_application_data.overlay_size

The size of the returned nmvt_application_data structure, and therefore the offset to the start of the next entry in the data buffer.

nmvt_application_data.application

Name of the registered application. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

nmvt_application_data.ms_vector_key_type

MS vector key accepted by the application. When the application registers for NMVT messages, it specifies which MS vector keys it will accept.

nmvt_application_data.conversion_required

Specifies whether the registered application requires incoming messages to be converted from NMVT to MDS_MU format. When the application registers for NMVT messages, it specifies whether this conversion is required. Possible values are:

AP_YES

Incoming messages are converted to MDS_MU format.

AP_NO

Incoming messages are not converted.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_APPLICATION_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the application parameter was

not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_NODE

QUERY_NODE returns information about the definition of a SNAplus2 node, and on its status if it is active.

VCB Structure

```
typedef struct query_node
{
  AP_UINT16        opcode;                  /* verb operation code        */
  unsigned char    reserv2;                 /* reserved                   */
  unsigned char    format;                  /* reserved                   */
  AP_UINT16        primary_rc;              /* primary return code        */
  AP_UINT32        secondary_rc;            /* secondary return code      */
  CP_CREATE_PARMS  cp_create_parms;         /* create parameters          */
  AP_UINT32        up_time;                 /* time since node started    */
  AP_UINT32        mem_size;                /* reserved                   */
  AP_UINT32        mem_used;                /* reserved                   */
  AP_UINT32        mem_warning_threshold;   /* reserved                   */
  AP_UINT32        mem_critical_threshold;  /* reserved                   */
  unsigned char    nn_functions_supported;  /* reserved                   */
  unsigned char    functions_supported;     /* functions supported        */
  unsigned char    en_functions_supported;  /* EN functions supported     */
  unsigned char    nn_status;               /* reserved                   */
  AP_UINT32        nn_frsn;                 /* reserved                   */
  AP_UINT32        nn_rsn;                  /* reserved                   */
  AP_UINT16        def_ls_good_xids;        /* Good XIDS for defined link */
                                            /*  stations                  */
  AP_UINT16        def_ls_bad_xids;         /* Bad XIDS for defined link  */
                                            /*  stations                  */
  AP_UINT16        dyn_ls_good_xids;        /* Good XIDS for dynamic link */
                                            /*  stations                  */
  AP_UINT16        dyn_ls_bad_xids;         /* Bad XIDS for dynamic link  */
                                            /*  stations                  */
  unsigned char    dlur_release_level;      /* Current DLUR release level */
  unsigned char    reserva[19];             /* reserved                   */
  unsigned char    fq_nn_server_name[17];   /* fully qualified NN server  */
                                            /*  name                      */
  AP_UINT32        current_isr_sessions;    /* reserved                   */
  unsigned char    reservb[30];             /* reserved                   */
 } QUERY_NODE;

typedef struct cp_create_parms
```

```
{
  AP_UINT16       crt_parms_len;          /* length of CP_CREATE_PARMS       */
  unsigned char   description[32];         /* resource description            */
  unsigned char   reserv1[16];            /* reserved                        */
  unsigned char   node_type;              /* node type                       */
  unsigned char   fqcp_name[17];          /* fully qualified CP name         */
  unsigned char   cp_alias[8];            /* CP alias                        */
  unsigned char   mode_to_cos_map_supp;   /* mode to COS mapping support     */
  unsigned char   mds_supported;          /* MDS and MS capabilities         */
  unsigned char   node_id[4];             /* node ID                         */
  AP_UINT16       max_locates;            /* maximum locates node can process*/
  AP_UINT16       dir_cache_size;         /* reserved                        */
  AP_UINT16       max_dir_entries;        /* maximum directory entries       */
                                          /*  (0 means unlimited)            */
  AP_UINT16       locate_timeout;         /* locate timeout in seconds       */
  unsigned char   reg_with_nn;            /* reserved                        */
  AP_UINT16       mds_send_alert_q_size;  /* size of MDS send alert queue    */
  AP_UINT16       cos_cache_size;         /* reserved                        */
  AP_UINT16       tree_cache_size;        /* reserved                        */
  AP_UINT16       tree_cache_use_limit;   /* reserved                        */
  AP_UINT16       max_tdm_nodes;          /* reserved                        */
  AP_UINT16       max_tdm_tgs;            /* reserved                        */
  AP_UINT32       max_isr_sessions;       /* reserved                        */
  AP_UINT32       isr_sessions_upper_threshold;  /* reserved                 */
  AP_UINT32       isr_sessions_lower_threshold;  /* reserved                 */
  AP_UINT16       isr_max_ru_size;        /* reserved                        */
  AP_UINT16       isr_rcv_pac_window;     /* reserved                        */
  unsigned char   store_endpt_rscvs;      /* endpoint RSCV storage           */
  unsigned char   store_isr_rscvs;        /* reserved                        */
  unsigned char   store_dlur_rscvs;       /* DLUR RSCV storage               */
  unsigned char   dlur_support;           /* is DLUR supported?              */
  unsigned char   pu_conc_support;        /* reserved                        */
  unsigned char   nn_rar;                 /* reserved                        */
  unsigned char   hpr_support;            /* reserved                        */
  unsigned char   mobile;                 /* reserved                        */
  unsigned char   discovery_support;      /* reserved                        */
  unsigned char   discovery_group_name[8];/* reserved                        */
  unsigned char   implicit_lu_0_to_3;     /* reserved                        */
  unsigned char   default_preference;     /* reserved                        */
  unsigned char   anynet_supported;       /* reserved                        */
#define MAX_LS_EXCEPTION_EVENTS 200
  AP_UINT16       max_ls_exception_events; /* max # exception entries        */
  unsigned char   reserv2[1];             /* reserved                        */
  unsigned char   max_compress_lvl;       /* Max compressson level (reserved)*/
  unsigned char   node_spec_data_len;     /* reserved                        */
  unsigned char   ptf[64];                /* program temporary fix array     */
```

```
} CP_CREATE_PARMS;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_NODE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

cp_create_parms.crt_parms_len

Length of create parameters structure.

cp_create_parms.description

A null-terminated text string describing the node, as specified in the definition of the node.

cp_create_parms.node_type

Type of node. Possible values are:

AP_END_NODE

AP_LEN_NODE

cp_create_parms.fqcp_name

Fully qualified name of the node. This is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

cp_create_parms.cp_alias

Locally used CP alias. This is an 8-byte ASCII string. All 8 bytes are significant.

cp_create_parms.mode_to_cos_map_supp

Specifies whether mode-to-COS mapping is supported by the node. For a LEN node, mode-to-COS mapping is not supported. Possible values are:

AP_YES

Mode-to-COS mapping is supported.

AP_NO

Mode-to-COS mapping is not supported.

cp_create_parms.mds_supported

Specifies whether Management Services supports Multiple Domain Support and MS Capabilities. Possible values are:

AP_YES

MDS is supported.

AP_NO

MDS is not supported.

cp_create_parms.node_id

Node identifier used in XID exchange. This is a 4-byte hexadecimal string.

cp_create_parms.max_locates

Maximum number of locates that the node can process.

cp_create_parms.dir_cache_size

Reserved

cp_create_parms.max_dir_entries

Maximum number of directory entries. Zero indicates no limit.

cp_create_parms.locate_timeout

Specifies the time in seconds before a network search will time out. Zero indicates no timeout.

cp_create_parms.reg_with_nn

End node only: Specifies whether to register the node's resources with the network node server when the node

is started. Possible values are:

AP_YES

Register resources with the NN. The end node's network node server will only forward directed locates to it.

AP_NO

Do not register resources. The network node server will forward all broadcast searches to the end node.

cp_create_parms.reg_with_cds

End node: Specifies whether the network node server is allowed to register end node resources with a Central Directory server. This field is ignored if *reg_with_nn* is set to AP_NO.

Possible values are:

AP_YES

Register resources with the CDS.

AP_NO

Do not register resources.

cp_create_parms.mds_send_alert_q_size

Size of the MDS send alert queue. If the number of queued alerts reaches this limit, SNAplus2 deletes the oldest alert on the queue.

cp_create_parms.cos_cache_size

This parameter is reserved.

cp_create_parms.tree_cache_size

This parameter is reserved.

cp_create_parms.tree_cache_use_limit

This parameter is reserved.

cp_create_parms.max_tdm_nodes

This parameter is reserved.

cp_create_parms.max_tdm_tgs

This parameter is reserved.

cp_create_parms.max_isr_sessions

This parameter is reserved.

cp_create_parms.isr_sessions_upper_threshold and *cp_create_parms.isr_sessions_lower_threshold*

This parameter is reserved.

cp_create_parms.isr_max_ru_size

This parameter is reserved.

cp_create_parms.isr_rcv_pac_window

This parameter is reserved.

cp_create_parms.store_endpt_rscvs

Specifies whether RSCVs should be stored for diagnostic purposes. Possible values are:

AP_YES

Store RSCVs.

AP_NO

Do not store RSCVs.

If this field is set to AP_YES, then an RSCV will be returned on the QUERY_SESSION verb. (Setting this value to AP_YES means an RSCV will be stored for each endpoint session. This extra storage can be up to 256 bytes per session.)

cp_create_parms.store_isr_rscvs

This parameter is reserved.

cp_create_parms.store_dlur_rscvs

Specifies whether RSCVs should be stored for diagnostic purposes (AP_YES or AP_NO). If this field is set to AP_YES, then an RSCV will be returned on the QUERY_DLUR_LU verb. (Setting this value to AP_YES means an RSCV will be stored for each PLU-SLU session. This extra storage can be up to 256 bytes per session.)

cp_create_parms.dlur_support

Specifies whether DLUR is supported. For a LEN node, this parameter is reserved. Possible values are:

AP_YES

AP_NO

cp_create_parms.pu_conc_support

Specifies whether PU concentration is supported (AP_YES or AP_NO).

cp_create_parms.nn_rar

Reserved.

cp_create_parms.hpr_support

This parameter is reserved.

cp_create_parms.max_ls_exception_events

The maximum number of LS exception events recorded by the node.

cp_create_parms.ptf

Array for configuring and controlling future program temporary fix (ptf) operation.

cp_create_parms.ptf[0]

REQDISCONT support. SNAplus2 normally uses REQDISCONT to deactivate limited resource host links that are no longer required by session traffic. This byte can be used to suppress use of REQDISCONT, or to modify the settings used on REQDISCONT requests sent by SNAplus2. This byte is set to one of the following values:

AP_NONE

Use the normal REQDISCONT support.

AP_SUPPRESS_REQDISCONT

Do not use REQDISCONT.

AP_OVERRIDE_REQDISCONT

Use a modified version of REQDISCONT support. This

value is combined with one or both of the following values, using a logical OR operation:

AP_REQDISCONT_TYPE

Use type “immediate” on REQDISCONT; if this value is not specified, SNAplus2 uses type “normal”.

AP_REQDISCONT_RECONTACT

Use type “immediate recontact” on REQDISCONT; if this value is not specified, SNAplus2 uses type “no immediate recontact”.

cp_create_parms.ptf[1]

ERP support. SNAplus2 normally processes an ACTPU(ERP) as an ERP; this resets the PU-SSCP session, but does not implicitly deactivate the subservient LU-SSCP and PLU-SLU sessions. SNA implementations may legally process ACTPU(ERP) as if it were ACTPU(cold), implicitly deactivating the subservient LU-SSCP and PLU-SLU sessions. This byte is set to one of the following values:

AP_NONE

Use the normal processing.

AP_OVERRIDE_ERP

Process all ACTPU requests as ACTPU(cold).

cp_create_parms.ptf[2]

BIS support. SNAplus2 normally uses the BIS protocol prior to deactivating a limited resource LU 6.2 session. This byte is set to one of the following values:

AP_NONE

Use the normal processing.

AP_SUPPRESS_BIS

Do not use the BIS protocol. Limited resource LU 6.2 sessions are deactivated immediately using UNBIND(cleanup).

up_time

Time (in hundredths of a second) since the node was started (or restarted). A value of zero indicates that the node is not running.

nn_functions_supported

Reserved.

functions_supported

Specifies the functions supported. This may be one or more of the following, combined using a logical OR.

AP_NEGOTIABLE_LS

AP_SEGMENT_REASSEMBLY

AP_BIND_REASSEMBLY

AP_PARALLEL_TGS

AP_CALL_IN

AP_ADAPTIVE_PACING

en_functions_supported

End node only: Specifies the end node functions supported. This may be one or more of the following, combined using a logical OR.

AP_SEGMENT_GENERATION

Node supports segment generation.

AP_MODE_TO_COS_MAP

Node supports mode name to COS name mapping.

AP_LOCATE_CDINIT

Node supports generation of locates and cross-domain initiate GDS variables for locating remote LUs.

AP_REG_WITH_NN

Node will register its LUs with the adjacent serving network node.

AP_REG_CHARS_WITH_NN

Node supports send register characteristics. If this function is supported, send registered names must also

be supported.

nn_status

Reserved.

nn_frsn

Reserved.

nn_rsn

Reserved.

def_ls_good_xids

Total number of successful XID exchanges that have occurred on all defined link stations since the node was last started.

def_ls_bad_xids

Total number of unsuccessful XID exchanges that have occurred on all defined link stations since the node was last started.

dyn_ls_good_xids

Total number of successful XID exchanges that have occurred on all dynamic link stations since the node was last started.

dyn_ls_bad_xids

Total number of unsuccessful XID exchanges that have occurred on all dynamic link stations since the node was last started.

dlur_release_level

Release level of the DLUR architecture supported by the node. This is set to the value `1` (the only release level of DLUR currently defined); future versions may incorporate later release levels of the DLUR architecture, and so may return different values.

fq_nn_server_name

End node only. Name of the network node server for the node.

current_isr_sessions

This parameter is reserved.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_NODE_ALL

QUERY_NODE_ALL returns information about nodes on the SNAplus2 LAN. This verb returns only each node's name and configuration file role, and does not provide detailed information about the node's configuration. The application can use QUERY_NODE for a particular node name to obtain detailed information about that node.

This verb must be issued with a null target handle.

VCB Structure

```
typedef struct query_node_all
{
   AP_UINT16       opcode;               /* verb operation code           */
   unsigned char   reserv2;              /* reserved                      */
   unsigned char   format;               /* reserved                      */
   AP_UINT16       primary_rc;           /* primary return code           */
   AP_UINT32       secondary_rc;         /* secondary return code         */
   unsigned char   *buf_ptr;             /* pointer to buffer             */
   AP_UINT32       buf_size;             /* buffer size                   */
   AP_UINT32       total_buf_size;       /* total buffer size required    */
   AP_UINT16       num_entries;          /* number of entries             */
   AP_UINT16       total_num_entries;    /* total number of entries       */
   unsigned char   list_options;         /* listing options               */
   unsigned char   reserv3;              /* reserved                      */
   unsigned char   node_name[64];        /* node name                     */
} QUERY_NODE_ALL;


typedef struct node_summary
{
   AP_UINT16       overlay_size;         /* size of returned entry        */
   unsigned char   node_name[64];        /* node name                     */
   unsigned char   config_role;          /* server's config file role     */
   unsigned char   reserv3[3];           /* reserved                      */
} NODE_SUMMARY;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_NODE_ALL

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of nodes for which data should be returned. To request data for a specific node rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list of nodes.

AP_LIST_INCLUSIVE

Start at the entry specified by the *node_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *node_name* parameter.

The list is not ordered by node name. However, the order remains the same for subsequent QUERY_NODE_ALL verbs, so the application can obtain a complete list in several sections by using multiple verbs in the normal way. For more information

about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

node_name

Name of the node to be used as an index into the list. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

This is an ASCII string of 1-64 characters, padded on the right with spaces if the name is shorter than 64 characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

node_summary.overlay_size

The size of the returned node_summary structure, and

therefore the offset to the start of the next entry in the data buffer.

node_summary.node_name

The name of the SNAplus2 node.

node_summary.config_role

The configuration file role of the server where the node is running. For more information about configuration file roles, refer to the *HP-UX SNAplus2 Administration Guide*. Possible values are:

AP_ROLE_MASTER

The server holds the master configuration file.

AP_ROLE_BACKUP

The server holds a backup configuration file.

AP_ROLE_NONE

The server does not share its copy of the configuration file.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_NODE_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *node_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_NODE_LIMITS

QUERY_NODE_LIMITS returns information about the functions that your SNAplus2 license allows you to use on a particular node. These are divided into two categories:

- Node options, which specify the SNAplus2 features that you can use
- Node limits, which specify the maximum numbers of LUs, sessions, and users that can use the node at any time.

The verb also returns information about the current usage of the parameters controlled by node limits.

VCB Structure

```
typedef struct query_node_limits
{
  AP_UINT16             opcode;        /* verb operation code               */
  unsigned char         reserv2;       /* reserved                          */
  unsigned char         format;        /* reserved                          */
  AP_UINT16             primary_rc;    /* primary return code               */
  AP_UINT32             secondary_rc;  /* secondary return code             */
  NODE_RESOURCE_LIMITS  max_limits;    /* max numbers of LUs/sessions/users  */
  NODE_RESOURCE_LIMITS  curr_usage;    /* current usage of LUs/sessions/users*/
  NODE_OPTIONS          node_options;  /* permitted functions               */
  unsigned char         reserv4[4];    /* reserved                          */
  NODE_RESOURCE_LIMITS  max_usage;     /* highest usage counts              */
} QUERY_NODE_LIMITS;


typedef struct node_resource_limits
{
   AP_INT32   num_lu_0_to_3_lus;         /* type 0 - 3 LUs                    */
   AP_INT32   num_lu_0_to_3_sessions;    /* type 0 - 3 sessions               */
   AP_INT32   num_local_lus;             /* local APPC LUs                    */
   AP_INT32   num_appc_sessions;         /* APPC sessions                     */
   AP_INT32   num_intermediate_sessions; /* ISR sessions                      */
   AP_INT32   num_user_sessions;         /* total APPC and LU 0 - 3 sessions  */
   AP_INT32   num_sna_users;             /* users                             */
   AP_INT32   num_total_sessions;        /* total licensed sessions           */
   AP_INT32   reserv1[2];                /* reserved                          */
} NODE_RESOURCE_LIMITS;
```

```
typedef struct node_options
{
   unsigned char   network_node;         /* is Network Node supported?       */
   unsigned char   end_node;             /* is End Node supported?           */
   unsigned char   len_node;             /* is LEN Node supported?           */
   unsigned char   dlur_support;         /* is DLUR supported?               */
   unsigned char   pu_conc_support;      /* is PU Conc supported?            */
   unsigned char   tn_server_support;    /* is TN Server supported?          */
   unsigned char   hpr_support;          /* level of HPR support             */
   unsigned char   back_level_client;    /* are back-level clients supported?*/
} NODE_OPTIONS;
```

Supplied Parameters

The application supplies the following parameter:

opcode AP_QUERY_NODE_LIMITS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

max_limits.num_lu_0_to_3_lus

The maximum number of type 0-3 LUs that your license allows you to define. A value of zero indicates that you cannot define any type 0-3 LUs; a value of -1 indicates no limit.

max_limits.num_lu_0_to_3_sessions

The maximum number of type 0-3 sessions that your license allows you to have active at any one time. A value of zero indicates that you cannot activate any type 0-3 sessions; a value of -1 indicates no limit.

max_limits.num_local_lus

The maximum number of local APPC LUs that your license allows you to define. A value of zero indicates that you cannot define any local APPC LUs; a value of

-1 indicates no limit.

max_limits.num_appc_sessions

The maximum number of APPC sessions that your license allows you to have active at any one time. A value of zero indicates that you cannot activate any APPC sessions; a value of -1 indicates no limit.

max_limits.num_intermediate_sessions

This parameter is reserved.

max_limits.num_user_sessions

The maximum total number of sessions (type 0-3 and APPC) that your license allows you to have active at any one time. A value of -1 indicates no limit.

max_limits.num_sna_users

The maximum number of users (3270 and 5250 emulation programs , RJE workstations, and APPC, CPI-C, and LUA applications) that your license allows you to have active at any one time. A value of -1 indicates no limit.

max_limits.num_total_sessions

The maximum total number of sessions of all types that your license allows you to have active at any one time. A value of -1 indicates that there is no limit on the number of users you can have active.

curr_usage.num_lu_0_to_3_lus

The number of type 0-3 LUs currently defined on this node.

curr_usage.num_lu_0_to_3_sessions

The number of type 0-3 sessions currently using this node.

curr_usage.num_local_lus

The number of local APPC LUs currently defined on this node.

curr_usage.num_appc_sessions

The number of APPC sessions currently using this node.

curr_usage.num_intermediate_sessions

This parameter is reserved.

curr_usage.num_user_sessions

The total number of sessions (type 0-3 and APPC) currently using this node.

curr_usage.num_sna_users

The number of users (3270 and 5250 emulation programs, RJE workstations, and APPC, CPI-C, and LUA applications) currently using this node. A value of `-1` indicates no limit.

curr_usage.num_total_sessions

The total number of sessions of all types that are currently active at this node.

max_usage.num_lu_0_to_3_lus

The highest number of type 0-3 LUs that have been defined at this node.

max_usage.num_lu_0_to_3_sessions

The highest number of type LU 0-3 sessions that have been active at any one time.

max_usage.num_local_lus

The highest number of local APPC LUs that have been defined.

max_usage.num_appc_sessions

The highest number of APPC sessions that have been active at any one time.

max_usage.num_intermediate_sessions

This parameter is reserved.

max_usage.num_user_sessions

The highest total number of type 0-3 and APPC sessions that have been active at any one time.

max_usage.num_sna_users

The highest number of users (3270 and 5250 emulation programs, RJE workstations, and APPC, CPI-C, and LUA applications) that have been active at any one time.

max_usage.num_total_sessions

The highest total number of sessions of all types that have been active at any one time.

node_options.network_node

This parameter is reserved.

node_options.end_node

Specifies whether your license allows you to define this node as an end node. Possible values are:

AP_YES

End node is supported.

AP_NO

End node is not supported.

node_options.len_node

Specifies whether your license allows you to define this node as a LEN node. Possible values are:

AP_YES

LEN node is supported.

AP_NO

LEN node is not supported.

node_options.dlur_support

This parameter is reserved.

Specifies whether your license allows you to use Dependent LU Requester (DLUR) on this node. Possible values are:

AP_YES

DLUR is supported.

AP_NO

DLUR is not supported.

node_options.pu_conc_support

Specifies whether your license allows you to use PU concentration on this node. Possible values are:

AP_YES

PU concentration is supported.

AP_NO

PU concentration is not supported.

node_options.tn_server_support

Specifies whether your license allows you to use TN server on this node. Possible values are:

AP_YES

TN server is supported.

AP_NO

TN server is not supported.

node_options.hpr_support

This parameter is reserved.

node_options.back_level_client

Specifies whether this server is running the software to support back-level clients; this software is used when you are in the process of migrating a client-server SNAplus2 system to a new release of the SNAplus2 software, so that different computers in the system are running different levels of the software. For more information about back-level client support, see the *HP-UX SNAplus2 Upgrade Guide*.

This parameter returns information about whether the software is currently running, not about license restrictions.

Possible values are:

AP_YES

The back-level client support software is running.

AP_NO

The back-level client support software is not running.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_PARTNER_LU

QUERY_PARTNER_LU returns information about partner LUs that a local LU is currently using, or has used. This verb returns information about usage of the partner LUs, not about their definition; use QUERY_PARTNER_LU_DEFINITION to obtain the definition of the partner LUs.

This verb can be used to obtain either summary or detailed information, about a specific LU or about multiple LUs, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_partner_lu
{
   AP_UINT16       opcode;              /* verb operation code             */
   unsigned char   reserv2;             /* reserved                        */
   unsigned char   format;              /* reserved                        */
   AP_UINT16       primary_rc;          /* primary return code             */
   AP_UINT32       secondary_rc;        /* secondary return code           */
   unsigned char   *buf_ptr;            /* pointer to buffer               */
   AP_UINT32       buf_size;            /* buffer size                     */
   AP_UINT32       total_buf_size;      /* total buffer size required      */
   AP_UINT16       num_entries;         /* number of entries               */
   AP_UINT16       total_num_entries;   /* total number of entries         */
   unsigned char   list_options;        /* listing options                 */
   unsigned char   reserv3;             /* reserved                        */
   unsigned char   lu_name[8];          /* LU name                         */
   unsigned char   lu_alias[8];         /* LU alias                        */
   unsigned char   plu_alias[8];        /* partner LU alias                */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name */
   unsigned char   active_sessions;     /* active sessions only filter     */
} QUERY_PARTNER_LU;

typedef struct plu_summary
{
   AP_UINT16       overlay_size;        /* size of returned entry          */
   unsigned char   plu_alias[8];        /* partner LU alias                */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name */
   unsigned char   reserv1;             /* reserved                        */
```

```
   unsigned char   description[32];       /* resource description            */
   unsigned char   reserv2[16];           /* reserved                        */
   AP_UINT16       act_sess_count;        /* currently active sessions count */
   unsigned char   partner_cp_name[17];   /* partner LU CP name              */
   unsigned char   partner_lu_located;    /* CP name resolved?               */
} PLU_SUMMARY;

typedef struct plu_detail
{
   AP_UINT16       overlay_size;          /* size of returned entry          */
   unsigned char   plu_alias[8];          /* partner LU alias                */
   unsigned char   fqplu_name[17];        /* fully qualified partner LU name */
   unsigned char   reserv1;               /* reserved                        */
   unsigned char   description[32];       /* resource description            */
   unsigned char   reserv2[16];           /* reserved                        */
   AP_UINT16       act_sess_count;        /* currently active sessions count */
   unsigned char   partner_cp_name[17];   /* partner LU CP name              */
   unsigned char   partner_lu_located;    /* CP name resolved?               */
   unsigned char   plu_un_name[8];        /* partner LU uninterpreted name   */
   unsigned char   parallel_sess_supp;    /* parallel sessions supported?    */
   unsigned char   conv_security;         /* conversation security           */
   AP_UINT16       max_mc_ll_send_size;   /* maximum send LL size for mapped */
                                          /*  conversations                  */
   unsigned char   implicit;              /* implicit or explicit entry      */
   unsigned char   security_details;      /* session security details        */
   unsigned char   duplex_support;        /* full-duplex support             */
   unsigned char   preference;            /* reserved                        */


  unsigned char    reserva[16];           /* reserved                        */
} PLU_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_PARTNER_LU

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and

should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

`AP_SUMMARY`

Summary information only.

`AP_DETAIL`

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

`AP_FIRST_IN_LIST`

Start at the first entry in the list of partner LUs associated with the specified local LU.

`AP_LIST_INCLUSIVE`

Start at the entry specified by the combination of local and partner LU names.

`AP_LIST_FROM_NEXT`

Start at the entry immediately following the entry specified by the combination of local and partner LU

names.

AP_LIST_BY_ALIAS

The list is returned in order of LU alias rather than LU name. This option is only valid if AP_FIRST_IN_LIST is also specified. (For AP_LIST_FROM_NEXT or AP_LIST_INCLUSIVE, the list is in order of LU alias or LU name, depending on which was specified as the index into the list.)

The combination of the local LU (*lu_name* or *lu_alias*) and partner LU (*plu_alias* or *fqplu_name*) specified is used as an index into the list of partner LUs if the *list_options* parameter is set to AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT.

The list is ordered by *fqplu_name*. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

lu_name

LU name of the local LU. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To indicate that the LU is identified by its LU alias instead of its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter.

lu_alias

LU alias of the local LU. This parameter is used only if the *lu_name* field is set to 8 binary zeros, and is ignored otherwise. The alias is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate the LU associated with the local CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored;

otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. To indicate that the LU is identified by its fully qualified name instead of its alias, set this parameter to 8 binary zeros and specify the LU name in the following parameter.

fqplu_name

17-byte fully qualified network name for the partner LU. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. This parameter is used only if the *plu_alias* field is set to 8 binary zeros, and is ignored otherwise.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

active_sessions

Specifies whether to return information only on partner LUs for which sessions are active, or on all partner LUs. Possible values are:

AP_YES

Return information only on partner LUs for which sessions are currently active.

AP_NO

Return information about all partner LUs for which sessions are active or have been active.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

plu_summary.overlay_size

The size of the returned plu_summary structure, and therefore the offset to the start of the next entry in the data buffer.

plu_summary.plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

plu_summary.fqplu_name

17-byte fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

plu_summary.description

A null-terminated text string describing the partner LU, as specified in the definition of the partner LU.

plu_summary.act_sess_count

Total number of active sessions between the local LU and the partner LU.

plu_summary.partner_cp_name

17-byte fully qualified network name for the CP associated with the partner LU. This parameter is not used if *partner_lu_located* below is set to AP_NO.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

plu_summary.partner_lu_located

Specifies whether the local node has located the CP where the partner LU is located. Possible values are:

AP_YES

The partner LU has been located. The *partner_cp_name* parameter contains the CP name of the partner LU.

AP_NO

The partner LU has not yet been located. The *partner_cp_name* parameter should not be checked.

plu_detail.overlay_size

The size of the returned plu_detail structure, and therefore the offset to the start of the next entry in the data buffer.

plu_detail.plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

plu_detail.fqplu_name

17-byte fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string

characters.

plu_detail.description

A null-terminated text string describing the partner LU, as specified in the definition of the partner LU.

plu_detail.act_sess_count

Total number of active sessions between the local LU and the partner LU.

plu_detail.partner_cp_name

17-byte fully qualified network name for the CP associated with the partner LU. This parameter is not used if *partner_lu_located* below is set to AP_NO.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

plu_detail.partner_lu_located

Specifies whether the local node has located the CP where the partner LU is located. Possible values are:

AP_YES

The partner LU has been located. The *partner_cp_name* parameter contains the CP name of the partner LU.

AP_NO

The partner LU has not yet been located. The *partner_cp_name* parameter should not be checked.

plu_detail.plu_un_name

Uninterpreted name of the partner LU. This is an 8-byte type-A EBCDIC character string.

plu_detail.parallel_sess_supp

Specifies whether parallel sessions are supported. Possible values are:

AP_YES

Parallel sessions are supported.

AP_NO

Parallel sessions are not supported.

plu_detail.conv_security

Specifies whether conversation security information can be sent to this partner LU. Possible values are:

AP_YES

Conversation security information supplied by a local TP is sent to the partner LU.

AP_NO

Conversation security information supplied by a local TP is not sent to the partner LU.

plu_detail.max_mc_ll_send_size

Maximum logical record size, in bytes, that can be sent to the partner LU. This may be in the range 1-32,767, or zero to indicate no limit (in which case the maximum is 32,767). Data records that are larger than this are broken down into several LL records before being sent to the partner LU.

plu_detail.implicit

Specifies whether the entry was created by an implicit or explicit definition. Possible values are:

AP_YES

The entry is an implicit entry.

AP_NO

The entry is an explicit entry.

plu_detail.security_details

Returns the conversation security support as negotiated on the BIND. Possible values are:

AP_CONVERSATION_LEVEL_SECURITY

Conversation security information will be accepted on requests to or from the partner LU to allocate a

conversation. The specific types of conversation security support are described by the following values.

AP_ALREADY_VERIFIED

Both the local and partner LU agree to accept Already Verified requests to allocate a conversation. An Already Verified request needs to carry only a user ID. It does not need to carry a password.

AP_PERSISTENT_VERIFICATION

Persistent Verification is supported on the session between the local and partner LUs. Once the initial request (carrying a user ID and usually a password) for a conversation has been verified, subsequent requests for a conversation need to carry only the user ID.

AP_PASSWORD_SUBSTITUTION

The local and partner LU support Password Substitution conversation security. When a request to allocate a conversation is issued, the request carries an encrypted form of the password. If Password Substitution is not supported, the password is carried in clear text (nonencrypted) format. If the session does not support Password Substitution, an Allocate or Send_Conversation with security type set to AP_PGM_STRONG will fail.

plu_detail.duplex_support

Returns the conversation duplex support as negotiated on the BIND. Possible values are:

AP_HALF_DUPLEX

Only half-duplex conversations are supported.

AP_FULL_DUPLEX

Both full-duplex and half-duplex sessions are supported. Expedited data is also supported.

AP_UNKNOWN

The conversation duplex support is not known because no sessions are active with the partner LU.

preference

This parameter is reserved.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LU_ALIAS

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_alias* parameter was not valid.

AP_INVALID_LU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *lu_name* parameter was not valid.

AP_INVALID_PLU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but one of the following conditions applies:

- The *fqplu_name* parameter does not match the name of any of this local LU's partners.
- No sessions have been active (since the node was last started) for the specified combination of local LU and partner LU.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_PARTNER_LU_DEFINITION

QUERY_PARTNER_LU_DEFINITION returns information about partner LUs for a local LU. This verb returns information about the definition of the LUs, not about their current usage; use QUERY_PARTNER_LU to obtain the usage information.

This verb can be used to obtain either summary or detailed information, about a specific LU or about multiple LUs, depending on the options used.

VCB Structure

```
typedef struct query_partner_lu_definition
{
   AP_UINT16       opcode;              /* verb operation code              */
   unsigned char   reserv2;             /* reserved                         */
   unsigned char   format;              /* reserved                         */
   AP_UINT16       primary_rc;          /* primary return code              */
   AP_UINT32       secondary_rc;        /* secondary return code            */
   unsigned char   *buf_ptr;            /* pointer to buffer                */
   AP_UINT32       buf_size;            /* buffer size                      */
   AP_UINT32       total_buf_size;      /* total buffer size required       */
   AP_UINT16       num_entries;         /* number of entries                */
   AP_UINT16       total_num_entries;   /* total number of entries          */
   unsigned char   list_options;        /* listing options                  */
   unsigned char   reserv3;             /* reserved                         */
   unsigned char   plu_alias[8];        /* partner LU alias                 */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name  */
} QUERY_PARTNER_LU_DEFINITION;

typedef struct partner_lu_def_summary
{
   AP_UINT16       overlay_size;        /* size of returned entry           */
   unsigned char   plu_alias[8];        /* partner LU alias                 */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name  */
   unsigned char   description[32];     /* resource description             */
   unsigned char   reserv1[16];         /* reserved                         */
} PARTNER_LU_DEF_SUMMARY;

typedef struct partner_lu_def_detail
{
```

```
   AP_UINT16       overlay_size;         /* size of returned entry           */
   unsigned char   plu_alias[8];         /* partner LU alias                 */
   unsigned char   fqplu_name[17];       /* fully qualified partner LU name  */
   unsigned char   reserv1;              /* reserved                         */
   PLU_CHARS       plu_chars;            /* partner LU characteristics       */
} PARTNER_LU_DEF_DETAIL;

typedef struct plu_chars
{
   unsigned char   fqplu_name[17];       /* fully qualified partner LU name  */
   unsigned char   plu_alias[8];         /* partner LU alias                 */
   unsigned char   description[32];      /* resource description             */
   unsigned char   reserv2[16];          /* reserved                         */
   unsigned char   plu_un_name[8];       /* partner LU uninterpreted name    */
   unsigned char   preference;           /* reserved                         */
   AP_UINT16       max_mc_ll_send_size;  /* maximum MC send LL size          */
   unsigned char   conv_security_ver;    /* already-verified security        */
                                         /*  supported?                      */
   unsigned char   parallel_sess_supp;   /* parallel sessions supported?     */
   unsigned char   reserv3[8];           /* reserved                         */
} PLU_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_PARTNER_LU_DEFINITION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of LUs for which data should be returned. To request data for a specific LU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *plu_alias* or *fqplu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *plu_alias* or *fqplu_name* parameter.

If AP_FIRST_IN_LIST is specified, you can also include the following option, using a logical OR operation:

AP_LIST_BY_ALIAS

The list is returned in order of LU alias rather than LU name. This option is only valid if AP_FIRST_IN_LIST is also specified. (For AP_LIST_FROM_NEXT or AP_LIST_INCLUSIVE, the list is in order of LU alias or LU name, depending on which was specified as the

index into the list.)

For more information about how the application can obtain specific entries from the list, see “List Options For QUERY_* Verbs”.

plu_alias

Partner LU alias. The name is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. To indicate that the partner LU is defined by its fully qualified name instead of its alias, set this parameter to 8 binary zeros and specify the name in the following parameter.

fqplu_name

Fully qualified name of the partner LU for which information is required, or the name to be used as an index into the list of LUs. If *list_options* is set to AP_FIRST_IN_LIST, this parameter is ignored; otherwise you must specify either the LU alias or the fully qualified LU name for the partner LU. This parameter is used only if the *plu_alias* parameter is set to zeros, and is ignored otherwise.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied

buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

partner_lu_def_summary.overlay_size

The size of the returned partner_lu_def_summary structure, and therefore the offset to the start of the next entry in the data buffer.

partner_lu_def_summary.plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

partner_lu_def_summary.fqplu_name

Fully qualified network name for the partner LU. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

partner_lu_def_summary.description

A null-terminated text string describing the partner LU, as specified in the definition of the partner LU.

partner_lu_def_detail.overlay_size

The size of the returned partner_lu_def_detail structure, and therefore the offset to the start of the

next entry in the data buffer.

partner_lu_def_detail.plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

partner_lu_def_detail.fqplu_name

Fully qualified network name for the partner LU. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

partner_lu_def_detail.plu_chars.fqplu_name

Fully qualified network name for the partner LU. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

partner_lu_def_detail.plu_chars.plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

partner_lu_def_detail.plu_chars.description

A null-terminated text string describing the partner LU, as specified in the definition of the partner LU.

partner_lu_def_detail.plu_chars.plu_un_name

Uninterpreted name of the partner LU. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

partner_lu_def_detail.plu_chars.preference

This parameter is reserved.

partner_lu_def_detail.plu_chars.max_mc_ll_send_size

Maximum logical record length, in bytes, that can be sent to the partner LU. This may be in the range 1-32,767, or zero to indicate no limit (in which case the

maximum is 32,767). Data records that are larger than this are broken down into several LL records before being sent to the partner LU.

partner_lu_def_detail.plu_chars.conv_security_ver

Specifies whether the partner LU is authorized to validate user IDs on behalf of local LUs; that is, whether the partner LU may set the already-verified indicator in an Attach request. Possible values are:

AP_YES

The partner LU is authorized to validate user IDs.

AP_NO

The partner LU is authorized to validate user IDs.

partner_lu_def_detail.plu_chars.parallel_sess_supp

Specifies whether parallel sessions are supported. Possible values are:

AP_YES

Parallel sessions are supported.

AP_NO

Parallel sessions are not supported.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_PLU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *plu_alias* or *fqplu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_PORT

QUERY_PORT returns a list of information about a node's ports. This information is structured as “determined data” (data gathered dynamically during execution) and “defined data” (the data supplied by the application on DEFINE_PORT).

This verb can be used to obtain either summary or detailed information, about a specific port or about multiple ports, depending on the options used.

VCB Structure

```
typedef struct query_port
{
  AP_UINT16        opcode;                /* verb operation code           */
  unsigned char    reserv2;               /* reserved                      */
  unsigned char    format;                /* reserved                      */
  AP_UINT16        primary_rc;            /* primary return code           */
  AP_UINT32        secondary_rc;          /* secondary return code         */
  unsigned char    *buf_ptr;              /* pointer to buffer             */
  AP_UINT32        buf_size;              /* buffer size                   */
  AP_UINT32        total_buf_size;        /* total buffer size required    */
  AP_UINT16        num_entries;           /* number of entries             */
  AP_UINT16        total_num_entries;     /* total number of entries       */
  unsigned char    list_options;          /* listing options               */
  unsigned char    reserv3;               /* reserved                      */
  unsigned char    port_name[8];          /* port name                     */
  unsigned char    dlc_name[8];           /* DLC name filter               */
} QUERY_PORT;

typedef struct port_summary
{
  AP_UINT16        overlay_size;          /* size of returned entry        */
  unsigned char    port_name[8];          /* port name                     */
  unsigned char    description[32];       /* resource description          */
  unsigned char    reserv2[16];           /* reserved                      */
  unsigned char    port_state;            /* port state                    */
  unsigned char    reserv1[1];            /* reserved                      */
  unsigned char    dlc_name[8];           /* name of DLC                   */
} PORT_SUMMARY;
```

```
typedef struct port_detail
{
  AP_UINT16        overlay_size;           /* size of returned entry        */
  unsigned char    port_name[8];           /* port name                     */
  unsigned char    reserv1[2];             /* reserved                      */
  PORT_DET_DATA    det_data;               /* determined data               */
  PORT_DEF_DATA    def_data;               /* defined data                  */
} PORT_DETAIL;

typedef struct port_det_data
{
  unsigned char    port_state;             /* port state                    */
  unsigned char    dlc_type;               /* DLC type                      */
  unsigned char    port_sim_rim;           /* port initialization options   */
  unsigned char    reserv1;                /* reserved                      */
  AP_UINT16        def_ls_good_xids;       /* number of successful XIDs     */
  AP_UINT16        def_ls_bad_xids;        /* number of unsuccessful XIDs   */
  AP_UINT16        dyn_ls_good_xids;       /* successful XIDs on dynamic    */
                                           /*  LS count                     */
  AP_UINT16        dyn_ls_bad_xids;        /* failed XIDs on dynamic LS     */
                                           /*  count                        */
  AP_UINT16        num_implicit_links;     /* number of implicit links      */
  unsigned char    neg_ls_supp;            /* negotiable?                   */
  unsigned char    reserva[17];            /* reserved                      */
} PORT_DET_DATA;

typedef struct port_def_data
{
  unsigned char    description[32];         /* resource description         */
  unsigned char    initially_active;       /* is the port initially active? */
  unsigned char    reserv2[15];            /* reserved                      */
  unsigned char    dlc_name[8];            /* DLC name associated with port */
  unsigned char    port_type;              /* port type                     */
  unsigned char    port_attributes[4];     /* port attributes               */
  unsigned char    reserv3[3];             /* reserved                      */
  AP_UINT32        port_number;            /* port number                   */
  AP_UINT16        max_rcv_btu_size;       /* max receive BTU size          */
  AP_UINT16        tot_link_act_lim;       /* total link activation limit   */
  AP_UINT16        inb_link_act_lim;       /* inbound link activation limit */
  AP_UINT16        out_link_act_lim;       /* outbound link activation limit*/
  unsigned char    ls_role;                /* initial link station role     */
  unsigned char    reserv1[15];            /* reserved                      */
  unsigned char    implicit_dspu_template; /* implicit dspu template        */
  AP_UINT16        implicit_ls_limit;      /* implicit ls limit             */
  unsigned char    reserv4;                /* reserved                      */
```

```
  unsigned char    implicit_dspu_services;  /* reserved                      */
  unsigned char    implicit_deact_timer;    /* deact timer for implicit LSs  */
  AP_UINT16        act_xid_exchange_limit;  /* activation XID exchange limit */
  AP_UINT16        nonact_xid_exchange_limit;     /* non-act. XID exchange   */
                                                  /*  limit                  */
  unsigned char    ls_xmit_rcv_cap;         /* LS transmit-receive capability*/
  unsigned char    max_ifrm_rcvd;           /* maximum number of I-frames    */
                                            /*  that can be received         */
  AP_UINT16        target_pacing_count;     /* target pacing count           */
  AP_UINT16        max_send_btu_size;       /* maximum send BTU size         */
  LINK_ADDRESS     dlc_data;                /* DLC data                      */
  LINK_ADDRESS     hpr_dlc_data;            /* reserved                      */
  unsigned char    implicit_cp_cp_sess_support;   /* implicit links allow    */
                                                  /*  CP-CP sessions         */
  unsigned char    implicit_limited_resource;     /* implicit links are      */
                                                  /*  limited resource       */
  unsigned char   implicit_hpr_support;     /* reserved                      */
  unsigned char   implicit_link_lvl_error;  /* reserved                      */
  unsigned char   retired1;                 /* reserved                      */
  TG_DEFINED_CHARS default_tg_chars;        /* default TG chars              */
  unsigned char   discovery_supported;      /* reserved                      */
  AP_UINT16       port_spec_data_len;       /* length of port specific data  */
  AP_UINT16       link_spec_data_len;       /* length of link specific data  */
} PORT_DEF_DATA;
```

For more details of the link_address structure and the port-specific and link-specific data, see “DEFINE_PORT” and “DEFINE_DLC”port_def_data. The data structure for the port-specific data follows the

structure, and the data structure for the link-specific data follows this; both structures are padded to start on a 4-byte boundary.

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_PORT

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and

should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of ports for which data should be returned. To request data for a specific port rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

`AP_SUMMARY`

Summary information only.

`AP_DETAIL`

Detailed information.

Combine this value using a logical `OR` operation with one of the following values:

`AP_FIRST_IN_LIST`

Start at the first entry in the list.

`AP_LIST_INCLUSIVE`

Start at the entry specified by the *port_name* parameter.

`AP_LIST_FROM_NEXT`

Start at the entry immediately following the entry specified by the *port_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

port_name

Name of port being queried. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

dlc_name

DLC name filter. To return information only on ports associated with a specific DLC, specify the DLC name. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To return information about all ports without filtering on the DLC name, set this parameter to 8 binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available

entries were returned.

Each entry in the data buffer consists of the following parameters:

port_summary.overlay_size

The size of the returned port_summary structure, and therefore the offset to the start of the next entry in the data buffer.

port_summary.port_name

Name of the port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

port_summary.description

A null-terminated text string describing the port, as specified in the definition of the port.

port_summary.port_state

Specifies the current state of the port. Possible values are:

AP_ACTIVE

The port is active.

AP_NOT_ACTIVE

The port is not active.

AP_PENDING_ACTIVE

START_PORT is in progress.

AP_PENDING_INACTIVE

STOP_PORT is in progress.

port_summary.dlc_name

Name of the DLC associated with this port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

port_detail.overlay_size

The size of the returned port_detail structure, and therefore the offset to the start of the next entry in the

data buffer.

port_detail.port_name

Name of the port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

port_detail.det_data.port_state

Specifies the current state of the port. Possible values are:

AP_ACTIVE

The port is active.

AP_NOT_ACTIVE

The port is not active.

AP_PENDING_ACTIVE

START_PORT is in progress.

AP_PENDING_INACTIVE

STOP_PORT is in progress.

port_detail.det_data.dlc_type

DLC type for the port. This is one of the following:

AP_SDLC

SDLC

AP_X25

QLLC

AP_TR

Token Ring

AP_ETHERNET

Ethernet

AP_FDDI

FDDI

port_detail.det_data.port_sim_rim

Specifies whether Set Initialization Mode (SIM) and Receive Initialization Mode (RIM) are supported. Possible values are:

AP_YES

SIM and RIM are supported.

AP_NO

SIM and RIM are not supported.

port_detail.det_data.def_ls_good_xids

Total number of successful XID exchanges that have occurred on all defined link stations on this port since the last time this port was started.

port_detail.det_data.def_ls_bad_xids

Total number of unsuccessful XID exchanges that have occurred on all defined link stations on this port since the last time this port was started.

port_detail.det_data.dyn_ls_good_xids

Total number of successful XID exchanges that have occurred on all dynamic link stations on this port since the last time this port was started.

port_detail.det_data.dyn_ls_bad_xids

Total number of unsuccessful XID exchanges that have occurred on all dynamic link stations on this port since the last time this port was started.

port_detail.det_data.num_implicit_links

Total number of implicit links currently active on this port. This includes dynamic links and implicit links created following use of Discovery. The number of such links allowed on this port is limited by the *implicit_ls_limit* parameter of port_def_data.

port_detail.det_data.neg_ls_supp

Support for negotiable link stations. Possible values are:

AP_YES

Link stations can be negotiated.

AP_NO

Link stations can not be negotiated.

port_detail.def_data.description

A null-terminated text string describing the port, as specified in the definition of the port.

port_detail.def_data.dlc_name

Name of the DLC associated with this port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

port_detail.def_data.port_type

The type of line used by the port.

For SDLC, the following values may be returned:

AP_PORT_SWITCHED

Switched line.

AP_PORT_NONSWITCHED

Nonswitched line.

For QLLC, this is set to AP_PORT_SWITCHED.

For Token Ring / Ethernet / FDDI , this is set to AP_PORT_SATF (shared access transport facility).

port_detail.def_data.port_attributes

This is a bit field. It can take the value AP_NO, or the following:

AP_RESOLVE_BY_LINK_ADDRESS

This value specifies that an attempt is made to resolve incoming calls by using the link address on CONNECT_IN before using the CP name (or node ID) carried on the received XID3 to resolve them. This is ignored if the *port_type* parameter is not set to AP_PORT_SWITCHED.

port_detail.def_data.port_number

Port number.

port_detail.def_data.max_rcv_btu_size

Maximum BTU size that can be received.

port_detail.def_data.tot_link_act_lim

Total link activation limit.

port_detail.def_data.inb_link_act_lim

Inbound link activation limit.

port_detail.def_data.out_link_act_lim

Outbound link activation limit.

port_detail.def_data.ls_role

Link station role.

For SDLC or QLLC, the following values may be returned:

AP_LS_PRI

Primary

AP_LS_SEC

Secondary

AP_LS_NEG

Negotiable

For Token Ring / Ethernet/ FDDI, this is set to AP_LS_NEG (negotiable).

port_detail.def_data.implicit_dspu_template

Specifies the DSPU template, defined with the DEFINE_DSPU_TEMPLATE verb, that will be used for definitions if the local node is to provide PU concentration for an implicit link activated on this port. If the template specified does not exist (or is already at its instance limit) when the link is activated, activation will fail. This is an 8-byte string in a locally displayable character set. All eight bytes are significant and must be set.

If the *def_data.implicit_dspu_services* parameter is not set to AP_PU_CONCENTRATION, this parameter is

reserved.

port_detail.def_data.implicit_ls_limit

The maximum number of implicit link stations which can be active on this port simultaneously, including dynamic links and links activated for Discovery. A value of zero indicates that there is no limit; a value of AP_NO_IMPLICIT_LINKS indicates that no implicit links are allowed.

port_detail.def_data.implicit_deact_timer

Limited resource link deactivation timer, in seconds.

If *implicit_limited_resource* is set to AP_INACTIVITY, then an implicit link is automatically deactivated if no data flows on the link for the duration of this timer.

port_detail.def_data.act_xid_exchange_limit

Activation XID exchange limit.

port_detail.def_data.nonact_xid_exchange_limit

Non-activation XID exchange limit.

port_detail.def_data.ls_xmit_rcv_cap

Specifies the link station transmit/receive capability. Possible values are:

AP_LS_TWS

Two-way simultaneous

AP_LS_TWA

Two-way alternating

port_detail.def_data.max_ifrm_rcvd

Maximum number of I-frames that can be received by local link stations before an acknowledgment is sent. Range: 1-127.

port_detail.def_data.target_pacing_count

Numeric value between 1 and 32,767 inclusive indicating the desired pacing window size. (The current version of SNAplus2 does not make use of this value.)

port_detail.def_data.max_send_btu_size

Maximum BTU size that can be sent.

port_detail.def_data.dlc_data

Port address. For more information, see "DEFINE_PORT".

def_data.implicit_cp_cp_sess_support

Specifies whether CP-CP sessions are permitted for implicit link stations using this port. Possible values are:

AP_YES

CP-CP sessions are permitted for implicit LSs.

AP_NO

CP-CP sessions are not permitted for implicit LSs.

def_data.implicit_limited_resource

Specifies whether implicit link stations off this port are defined as limited resources. Possible values are:

AP_NO

Implicit links are not limited resources, and will not be deactivated automatically.

AP_NO_SESSIONS

Implicit links are limited resources, and will be deactivated automatically when no active sessions are using them.

AP_INACTIVITY

Implicit links are limited resources, and will be deactivated automatically when no active sessions are using them or when no data has flowed for the time period specified by the *implicit_deact_timer* field.

def_data.implicit_hpr_support

This parameter is reserved.

def_data.implicit_link_lvl_error

This parameter is reserved.

def_data.default_tg_chars

Default TG characteristics. These are used for implicit link stations using this port, and as the default TG characteristics for defined link stations that do not have TG characteristics explicitly defined. For details of these parameters, see "DEFINE_LS".

port_detail.def_data.port_spec_data_len

Unpadded length, in bytes, of the port-specific data. The data structure for this data follows the port_def_data structure, but is padded to start on a 4-byte boundary. For more details of the port-specific data, see "DEFINE_PORT".

port_detail.def_data.link_spec_data_len

Data passed unchanged to link station component during initialization. The data structure for the link-specific data follows the data structure for the port-specific data, but is padded to start on a 4-byte boundary. For more details of the link-specific data, see "DEFINE_PORT".

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PORT_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *port_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all

NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_PU

QUERY_PU returns information about local PUs and the links associated with them. This verb can be used to obtain information about a specific PU or about multiple PUs, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_pu
{
   AP_UINT16       opcode;              /* verb operation code            */
   unsigned char   reserv2;             /* reserved                       */
   unsigned char   format;              /* reserved                       */
   AP_UINT16       primary_rc;          /* primary return code            */
   AP_UINT32       secondary_rc;        /* secondary return code          */
   unsigned char   *buf_ptr;            /* pointer to buffer              */
   AP_UINT32       buf_size;            /* buffer size                    */
   AP_UINT32       total_buf_size;      /* total buffer size required     */
   AP_UINT16       num_entries;         /* number of entries              */
   AP_UINT16       total_num_entries;   /* total number of entries        */
   unsigned char   list_options;        /* listing options                */
   unsigned char   reserv3;             /* reserved                       */
   unsigned char   pu_name[8];          /* PU name                        */
   unsigned char   host_attachment;     /* host attachment filter         */
} QUERY_PU;

typedef struct pu_data
{
   AP_UINT16       overlay_size;        /* size of returned entry         */
   unsigned char   pu_name[8];          /* PU name                        */
   unsigned char   description[32];     /* resource description           */
   unsigned char   reserv1[16];         /* reserved                       */
   unsigned char   ls_name[8];          /* LS name                        */
   unsigned char   pu_sscp_sess_active; /* Is PU-SSCP session active      */
   unsigned char   host_attachment;     /* Host attachment                */
   unsigned char   reserv1[2];          /* reserved                       */
   SESSION_STATS   pu_sscp_stats;       /* PU-SSCP session statistics     */
   unsigned char   sscp_id[6];          /* SSCP ID                        */
   unsigned char   reserva[14];         /* reserved                       */
} PU_DATA;
```

```
typedef struct session_stats
{
   AP_UINT16       rcv_ru_size;           /* session receive RU size         */
   AP_UINT16       send_ru_size;          /* session send RU size            */
   AP_UINT16       max_send_btu_size;     /* maximum send BTU size           */
   AP_UINT16       max_rcv_btu_size;      /* maximum rcv BTU size            */
   AP_UINT16       max_send_pac_win;      /* maximum send pacing window size */
   AP_UINT16       cur_send_pac_win;      /* current send pacing window size */
   AP_UINT16       max_rcv_pac_win;       /* maximum receive pacing window   */
                                          /*  size                           */
   AP_UINT16       cur_rcv_pac_win;       /* current receive pacing window   */
                                          /*  size                           */
   AP_UINT32       send_data_frames;      /* number of data frames sent      */
   AP_UINT32       send_fmd_data_frames;  /* num fmd data frames sent        */
   AP_UINT32       send_data_bytes;       /* number of data bytes sent       */
   AP_UINT32       rcv_data_frames;       /* number of data frames received  */
   AP_UINT32       rcv_fmd_data_frames;   /* num fmd data frames received    */
   AP_UINT32       rcv_data_bytes;        /* number of data bytes received   */
   unsigned char   sidh;                  /* session ID high byte (from      */
                                          /*  LFSID)                         */
   unsigned char   sidl;                  /* session ID low byte (from LFSID)*/
   unsigned char   odai;                  /* ODAI bit set                    */
   unsigned char   ls_name[8];            /* Link station name               */
   unsigned char   pacing_type;           /* type of pacing in use           */
} SESSION_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_PU

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of PUs for which data should be returned. To request data for a specific PU rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *pu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *pu_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

pu_name

Name of the PU for which information is required, or the name to be used as an index into the list of PUs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

host_attachment

Specifies whether to filter the returned information by whether the PUs are attached to the host directly or using DLUR. Possible values are:

AP_DIRECT_ATTACHED

Return information only on PUs directly attached to the host system.

AP_DLUR_ATTACHED

Return information only on PUs supported by DLUR.

AP_NONE

Return information about all PUs regardless of host attachment.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

pu_data.overlay_size

The size of the returned pu_data structure, and therefore the offset to the start of the next entry in the data buffer.

pu_data.pu_name

PU Name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

pu_data.description

A null-terminated text string describing the PU, as specified in the definition of the LS or of the internal PU.

pu_data.ls_name

Name of the link station associated with this PU. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

pu_data.pu_sscp_sess_active

Specifies whether the PU-SSCP session is active. Possible values are:

AP_YES

The PU-SSCP session is active.

AP_NO

The PU-SSCP session is inactive.

pu_data.host_attachment

Local PU host attachment type.

Possible values are:

AP_DIRECT_ATTACHED

PU is directly attached to the host system.

AP_DLUR_ATTACHED

PU is supported by DLUR.

pu_data.pu_sscp_stats.rcv_ru_size

Reserved (always set to zero).

pu_data.pu_sscp_stats.send_ru_size

Reserved (always set to zero).

pu_data.pu_sscp_stats.max_send_btu_size

Maximum BTU size that can be sent.

pu_data.pu_sscp_stats.max_rcv_btu_size

Maximum BTU size that can be received.

pu_data.pu_sscp_stats.max_send_pac_win

Reserved (always set to zero).

pu_data.pu_sscp_stats.cur_send_pac_win

Reserved (always set to zero).

pu_data.pu_sscp_stats.max_rcv_pac_win

Reserved (always set to zero).

pu_data.pu_sscp_stats.cur_rcv_pac_win

Reserved (always set to zero).

pu_data.pu_sscp_stats.send_data_frames

Number of normal flow data frames sent.

pu_data.pu_sscp_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

pu_data.pu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

pu_data.pu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

ppu_data.pu_sscp_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

pu_data.pu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

pu_data.pu_sscp_stats.sidh

Session ID high byte.

pu_data.pu_sscp_stats.sidl

Session ID low byte.

pu_data.pu_sscp_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station.

pu_data.pu_sscp_stats.ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters.

pu_data.pu_sscp_stats.pacing_type

The type of receive pacing in use on the PU-SSCP session. This parameter is set to AP_NONE.

pu_data.sscp_id

For dependent LU sessions, this parameter is the SSCP ID received in the ACTPU from the host for the PU to which the local LU is mapped. For independent LU sessions, this parameter is set to 0 (zero). This value is an array of six bytes displayed as hexadecimal values.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PU_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *pu_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_PU_TYPE

The PU specified by the *pu_name* parameter is a remote PU and not a local PU.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_RCF_ACCESS

QUERY_RCF_ACCESS returns information about the permitted access to the SNAplus2 Remote Command Facility (RCF): the user ID used to run HP-UX Command Facility (UCF) commands, and the restrictions on which administration commands can be issued using the Service Point Command Facility (SPCF). This information was previously set up using DEFINE_RCF_ACCESS. For more information about SPCF and UCF, see the *HP-UX SNAplus2 Administration Guide*.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct query_rcf_access
{
   AP_UINT16         opcode;                 /* Verb operation code      */
   unsigned char     reserv2;                /* reserved                 */
   unsigned char     format;                 /* reserved                 */
   AP_UINT16         primary_rc;             /* primary return code      */
   AP_UINT32         secondary_rc;           /* secondary return code    */
   unsigned char     ucf_username[32];       /* UCF username             */
   AP_UINT32         spcf_permissions;       /* SPCF permissions         */
} QUERY_RCF_ACCESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_RCF_ACCESS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

ucf_username

Specifies the HP-UX user name of the UCF user. This

parameter is a null-terminated ASCII string.

All UCF commands will be run using this user's user ID, using the default shell and access permissions defined for this user.

If this parameter is set to a null string, this indicates that UCF access is prohibited.

spcf_permissions

Specifies the types of SNAplus2 administration commands that can be accessed using SPCF. This is set to AP_NONE to indicate that SPCF access is prohibited, or to one or more of the following values (combined using a logical OR):

AP_ALLOW_QUERY_LOCAL

QUERY_* verbs are permitted.

AP_ALLOW_DEFINE_LOCAL

DEFINE_*, SET_*, DELETE_*, ADD_*, and REMOVE_* verbs, and also INIT_NODE, are permitted.

AP_ALLOW_ACTION_LOCAL

“Action” verbs are permitted: START_*, STOP_*, ACTIVATE_*, DEACTIVATE_*, and also APING, INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, and RESET_SESSION_LIMIT.

AP_ALLOW_QUERY_REMOTE

The QUERY_* verbs are allowed to provide access to a remote SNAplus2 node.

AP_ALLOW_DEFINE_REMOTE

The DEFINE_*, SET_*, DELETE_*, ADD_*, REMOVE_*, and INIT_NODE verbs are allowed to provide access to a remote SNAplus2 node.

AP_ALLOW_ACTION_REMOTE

The START_*, STOP_*, ACTIVATE_*, DEACTIVATE_*, APING,

INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, and RESET_SESSION_LIMIT verbs are allowed to provide access to a remote SNAplus2 node.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_RJE_WKSTN

QUERY_RJE_WKSTN returns information about usage of SNAplus2 RJE workstations. It can return either summary or detailed information, about a single workstation or multiple workstations, depending on the options used. This verb returns information about the current usage of the workstation, not about its definition; use QUERY_RJE_WKSTN_DEF to obtain information about the configuration file definition.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_rje_wkstn
{
   AP_UINT16        opcode;                 /* verb operation code          */
   unsigned char    reserv2;                /* reserved                     */
   unsigned char    format;                 /* reserved                     */
   AP_UINT16        primary_rc;             /* primary return code          */
   AP_UINT32        secondary_rc;           /* secondary return code        */
   unsigned char    *buf_ptr;               /* pointer to buffer            */
   AP_UINT32        buf_size;               /* buffer size                  */
   AP_UINT32        total_buf_size;         /* total buffer size required   */
   AP_UINT16        num_entries;            /* number of entries            */
   AP_UINT16        total_num_entries;      /* total number of entries      */
   unsigned char    list_options;           /* listing options              */
   unsigned char    reserv3;                /* reserved                     */
   unsigned char    workstation_name[4];    /* workstation name             */
   unsigned char    system_name[64];        /* computer name                */
} QUERY_RJE_WKSTN;

typedef struct rje_wkstn_summary
{
   AP_UINT16        overlay_size;           /* size of returned entry       */
   unsigned char    workstation_name[4];    /* workstation name             */
   unsigned char    system_name[64];        /* computer name                */
   AP_UINT32        wkstn_pid;              /* process ID                   */
} RJE_WKSTN_SUMMARY;

typedef struct rje_wkstn_detail
{
```

```
   AP_UINT16        overlay_size;            /* size of returned entry       */
   unsigned char    workstation_name[4];     /* workstation name             */
   unsigned char    system_name[64];         /* computer name                */
   AP_UINT32        wkstn_pid;               /* process ID                   */
   AP_UINT32        reserv1;                 /* reserved                     */
   AP_UINT32        wkstn_uid;               /* user ID                      */
   AP_UINT32        wkstn_gid;               /* group ID                     */
   unsigned char    wkstn_uname[32];         /* user name                    */
   unsigned char    wkstn_gname[32];         /* group name                   */
   AP_UINT32        wkstn_session_count;     /* count of RJE LU sessions     */
   AP_UINT32        wkstn_start_time;        /* time workstation was started */
} RJE_WKSTN_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_RJE_WKSTN

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of workstations for which data should be returned. To request data for a specific workstation rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *workstation_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *workstation_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

workstation_name

The name of the workstation for which information is required, or the name to be used as an index into the list of workstations. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

system_name

Computer name for which RJE workstation information is required. This is an ASCII string, padded on the right with spaces.

To list only information about workstations running on a particular computer, specify the computer name. To obtain a complete list for all computers, set this field to binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned.

Each entry in the data buffer consists of the following:

rje_wkstn_summary.overlay_size

The size of the returned rje_wkstn_summary structure, and therefore the offset to the start of the next entry in the data buffer.

rje_wkstn_summary.workstation_name

The name of the RJE workstation.

rje_wkstn_summary.system_name

The computer name on which the workstation is running. This is an ASCII string, padded on the right with spaces.

rje_wkstn_summary.wkstn_pid

The process ID of the workstation.

rje_wkstn_detail.overlay_size

The size of the returned rje_wkstn_detail structure,

and therefore the offset to the start of the next entry in the data buffer.

rje_wkstn_detail.workstation_name

The name of the RJE workstation.

rje_wkstn_detail.system_name

The computer name on which the workstation is running. This is an ASCII string, padded on the right with spaces.

rje_wkstn_detail.wkstn_pid

The process ID of the workstation.

rje_wkstn_detail.wkstn_uid

The HP-UX user ID with which the workstation is running.

rje_wkstn_detail.wkstn_gid

The HP-UX group ID with which the workstation is running.

rje_wkstn_detail.wkstn_uname

The HP-UX user name with which the workstation is running. This is an ASCII string, padded on the right with spaces.

rje_wkstn_detail.wkstn_gname

The HP-UX group name with which the workstation is running. This is an ASCII string, padded on the right with spaces.

rje_wkstn_detail.wkstn_session_count

The number of LU sessions currently active for this workstation.

rje_wkstn_detail.wkstn_start_time

The time at which the workstation was started (this may be earlier than the start time for the first LU session). This value is specified as “seconds since epoch” (the number of seconds since the start of the year 1970).

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_WORKSTATION

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the specified name, but the *workstation_name* or *system_name* parameter was not set to a valid value.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_RJE_WKSTN_DEF

QUERY_RJE_WKSTN_DEF returns information about RJE workstations. It can return either summary or detailed information, about a single workstation or multiple workstations, depending on the options used. This verb returns information about the definition of the workstation, not about its current usage; use QUERY_RJE_WKSTN to obtain information about current usage.

This verb must be issued to the domain configuration file.

VCB Structure

```
typedef struct query_rje_wkstn_def
{
   AP_UINT16           opcode;              /* verb operation code          */
   unsigned char       reserv2;             /* reserved                     */
   unsigned char       format;              /* reserved                     */
   AP_UINT16           primary_rc;          /* primary return code          */
   AP_UINT32           secondary_rc;        /* secondary return code        */
   unsigned char       *buf_ptr;            /* pointer to buffer            */
   AP_UINT32           buf_size;            /* buffer size                  */
   AP_UINT32           total_buf_size;      /* total buffer size required   */
   AP_UINT16           num_entries;         /* number of entries            */
   AP_UINT16           total_num_entries;   /* total number of entries      */
                                            /* - includes partial entries   */
   unsigned char       list_options;        /* listing options              */
   unsigned char       reserv3;             /* reserved                     */
   unsigned char       workstation_name[4]; /* RJE workstation name         */
} QUERY_RJE_WKSTN_DEF;


   typedef struct rje_wkstn_def_summary
{
   AP_UINT16           overlay_size;        /* size of returned entry       */
   unsigned char       workstation_name[4]; /* RJE workstation name         */
} RJE_WKSTN_DEF_SUMMARY;


typedef struct rje_wkstn_def_detail
{
   AP_UINT16           overlay_size;        /* size of returned entry       */
   unsigned char       workstation_name[4]; /* workstation name             */
   unsigned char       reserv3[8];          /* Reserved                     */
```

```
   RJE_WKSTN_DEF_DATA  def_data;
} RJE_WKSTN_DEF_DETAIL;


typedef struct rje_wkstn_def_data
{
   unsigned char       description[32];      /* Description - null terminated*/
   unsigned char       reserv1[16];          /* reserved                     */
   unsigned char       primary_user[32];     /* primary user name            */
   unsigned char       group_name[32];       /* user's group name            */
   unsigned char       system_name[64];      /* computer where workstation   */
                                             /*  runs                        */
   AP_UINT16           num_lus;              /* count field for the number   */
                                             /*  of LUs                      */
   unsigned char       reserv4[18];          /* Reserved                     */
   unsigned char       lu_name[5][8];        /* LU or LU Group used by       */
                                             /*  workstation (up to 5)       */
 RJE_WKSTN_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_RJE_WKSTN_DEF

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof() function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of workstations for which data should be returned. To request data for a specific workstation rather than a range, specify the value 1. To

return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the supplied workstation name.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the supplied workstation name.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

workstation_name

The name of the workstation for which information is required, or the name to be used as an index into the list of workstations. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

num_entries

The number of entries actually returned.

Each entry in the data buffer consists of the following:

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

rje_wkstn_def_summary.overlay_size

The size of the returned rje_wkstn_def_summary structure, and therefore the offset to the start of the next entry in the data buffer.

rje_wkstn_def_summary.workstation_name

The name of the workstation.

rje_wkstn_def_detail.overlay_size

The size of the returned rje_wkstn_def_detail structure, and therefore the offset to the start of the next entry in the data buffer.

rje_wkstn_def_detail.workstation_name

The name of the workstation.

rje_wkstn_def_detail.def_data

The details of the workstation, as defined in the

configuration. The format of this information is the same as for the DEFINE_RJE_WKSTN verb.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_WORKSTATION

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the specified name, but the *workstation_name* parameter did not match any defined RJE workstation name.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_SECURITY_ACCESS_LIST

QUERY_SECURITY_ACCESS_LIST returns information about security access lists defined in a SNAplus2 configuration file. It can return information about a single list or multiple lists, depending on the options used

VCB Structure

```
typedef struct query_security_access_list
{
   AP_UINT16         opcode;               /* verb operation code           */
   unsigned char     reserv2;              /* reserved                      */
   unsigned char     format;               /* reserved                      */
   AP_UINT16         primary_rc;           /* primary return code           */
   AP_UINT32         secondary_rc;         /* secondary return code         */
   unsigned char     *buf_ptr;             /* pointer to buffer             */
   AP_UINT32         buf_size;             /* buffer size                   */
   AP_UINT32         total_buf_size;       /* total buffer size required    */
   AP_UINT16         num_entries;          /* number of entries             */
   AP_UINT16         total_num_entrieS;    /* total number of entries       */
   unsigned cha      list_options;         /* listing options               */
   unsigned char     reserv3;              /* reserved                      */
   unsigned char     list_name[14];        /* Security Access List name     */
   unsigned char     user_name[10];        /* user name                     */
   AP_UINT32         num_init_users;       /* number of users for first     */
                                           /*  list when starting in middle */
   AP_UINT32         num_last_users;       /* number of users on last       */
                                           /*  overlay if last list is      */
                                           /*  incomplete                   */
   unsigned char     last_list_incomplete; /* set to AP_YES if user data    */
                                           /*  for last list is incomplete  */
} QUERY_SECURITY_ACCESS_LIST;


typedef struct security_access_detail
{
   AP_UINT16         overlay_size;       /* size of returned entry         */
   unsigned char     list_name[14];      /* list name                      */
   unsigned char     reserv1[2];         /* reserved                       */
   AP_UINT32         num_filtered_users; /* number of users returned       */
   SECURITY_LIST_DEF def_data;           /* list definition                */
```

```
} SECURITY_ACCESS_DETAIL;


typedef struct security_list_def
{
   unsigned char     description[32];    /* description                    */
   unsigned char     reserv3[16];        /* reserved                       */
   AP_UINT32         num_users;          /* number of users in list        */
   unsigned char     reserv2[16];        /* reserved                       */
} SECURITY_LIST_DEF;


typedef struct security_user_data
{
   AP_UINT16         sub_overlay_size;   /* reserved                       */
   unsigned char     user_name[10];      /* user name                      */
} SECURITY_USER_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_QUERY_SECURITY_ACCESS_LIST

buf_ptr A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size Size of the supplied data buffer.

num_entries Maximum number of security access lists for which data should be returned. This number includes partial security access list entries (for which a user name is specified, so that the returned data does not include the first user name in the list).

To request data for a specific security access list rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options The position in the list from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first user name for the first security access list.

AP_LIST_INCLUSIVE

Start at the entry specified by the supplied security access list name and user name, or start at the first user name for the specified security access list if no user name is specified.

AP_LIST_FROM_NEXT

If a user name is specified, start at the user immediately following the specified user. If no user name is specified, start at the first user for the specified security access list.

The list is ordered by security access list name, and then by user name within each security access list. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

list_name The name of the security access list for which information is required, or the name to be used as an index into the list of security access lists. This parameter is ignored if list_options is set to AP_FIRST_IN_LIST. The name is an ASCII string of 1 - 14 characters, padded on the right with spaces if the name is shorter than 14 characters.

user_name To return information starting with a specific user name for the specified security access list, set this parameter to the user name. To return information starting at the first user name for the specified security access list, set this parameter to 10 binary zeros.

Returned parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned. The last entry may be incomplete. This is indicated by the *last_list_incomplete* parameter.

num_init_users

If the *user_name* parameter was set to a nonzero value, so that the information for the first security access list in the returned data does not start with the first user in that list, this parameter indicates the number of user name structures for this list that are included in the returned data. Otherwise, this parameter is not used.

num_last_users

If the *last_list_incomplete* parameter indicates that the data for the last list is incomplete, this parameter indicates the number of user name structures for this list that are included in the returned data. (The *num_filtered_users* parameter returned for this list indicates the total number of user name structures that are available). Otherwise, this parameter is not used.

last_list_incomplete

Specified whether the information for the last security access list is incomplete. Possible values are:

AP_YES

The complete data for the last security access list was too large to fit in the data buffer. At least one user

name structure is included, but there are further user name structures that are not included in the data buffer. The *num_last_users* parameter indicates how many user name structures have been returned. The application can issue further verbs to obtain the remaining data.

AP_NO

The data for the last list is complete.

Each entry in the data buffer consists of the following.

security_access_detail.list_name

The name of the security access list. This is an ASCII string of 1 - 14 characters.

security_access_detail.num_filtered_users

The total number of user names in this security access list.

security_access_detail.def_data.description

A null-terminated text string describing the security access list, as specified in the definition of the list.

security_access_detail.def_data.num_users

The total number of users in the security access list.

If this is the last list in the data buffer, and the *last_list_incomplete* parameter is set to AP_YES, the total number of user name structures returned for this list will be as specified by the *num_last_users* parameter; this will be less than *num_users*.

For each user name in the list, a *security_user_data* structure is returned with the follow information:

user_name

Name of the user. This is a user ID defined using the DEFINE_USERID_PASSWORD verb.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LIST_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *list_name* parameter did not match the names of any defined security access list.

AP_INVALID_USER_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *user_name* parameter did not match a user name defined for the specified security access list.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_SESSION

QUERY_SESSION returns list information about sessions for a particular local LU.

This verb can be used to obtain either summary or detailed information, about a specific session or a range of sessions, depending on the options used.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_session
{
   AP_UINT16       opcode;              /* verb operation code              */
   unsigned char   reserv2;             /* reserved                         */
   unsigned char   format;              /* reserved                         */
   AP_UINT16       primary_rc;          /* primary return code              */
   AP_UINT32       secondary_rc;        /* secondary return code            */
   unsigned char   *buf_ptr;            /* pointer to buffer                */
   AP_UINT32       buf_size;            /* buffer size                      */
   AP_UINT32       total_buf_size;      /* total buffer size required       */
   AP_UINT16       num_entries;         /* number of entries                */
   AP_UINT16       total_num_entries;   /* total number of entries          */
   unsigned char   list_options;        /* listing options                  */
   unsigned char   reserv3;             /* reserved                         */
   unsigned char   lu_name[8];          /* LU name                          */
   unsigned char   lu_alias[8];         /* LU alias                         */
   unsigned char   plu_alias[8];        /* partner LU alias                 */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name  */
   unsigned char   mode_name[8];        /* mode name                        */
   unsigned char   session_id[8];       /* session ID                       */
} QUERY_SESSION;


typedef struct session_summary
{
   AP_UINT16       overlay_size;        /* size of returned entry           */
   unsigned char   plu_alias[8];        /* partner LU alias                 */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name  */
   unsigned char   reserv3[3];          /* reserved                         */
   unsigned char   mode_name[8];        /* mode name                        */
   unsigned char   session_id[8];       /* session ID                       */
```

```
   FQPCID          fqpcid;              /* fully qualified procedure        */
                                        /*  correlator ID                   */
} SESSION_SUMMARY;


typedef struct session_detail
{
   AP_UINT16       overlay_size;        /* size of returned entry           */
   unsigned char   plu_alias[8];        /* partner LU alias                 */
   unsigned char   fqplu_name[17];      /* fully qualified partner LU name  */
   unsigned char   reserv3[3];          /* reserved                         */
   unsigned char   mode_name[8];        /* mode name                        */
   unsigned char   session_id[8];       /* session ID                       */
   FQPCID          fqpcid;              /* fully qualified procedure        */
                                        /*  correlator ID                   */
   unsigned char   cos_name[8];         /* Class of Service name            */
   unsigned char   trans_pri;           /* Transmission priority:           */
   unsigned char   ltd_res;             /* Session spans a limited resource */
   unsigned char   polarity;            /* Session polarity                 */
   unsigned char   contention;          /* Session contention               */
   SESSION_STATS   sess_stats;          /* Session statistics               */
   unsigned char   duplex_support       /* full-duplex support              */
   unsigned char   sscp_id              /* SSCP ID of host                  */
   unsigned char   reserva;             /* reserved                         */

   AP_UINT32       session_start_time   /* start time of the session        */
   AP_UINT16       session_timeout      /* session timeout                  */
   unsigned char   reservb[7];          /* reserved                         */
   unsigned char   plu_slu_comp_lvl     /* PLU to SLU compression level     */
   unsigned char   slu_plu_comp_lvl     /* SLU to PLU compression level     */
   unsigned char   rscv_len;            /* Length of following RSCV         */
} SESSION_DETAIL;
```

The session detail structure may be followed by a Route Selection Control Vector (RSCV) as defined by SNA Formats. This control vector defines the session route through the network and is carried on the BIND. This RSCV is included only if the node's configuration (specified using DEFINE_NODE) indicates that endpoint RSCVs should be stored.

```
typedef struct fqpcid
{
   unsigned char   pcid[8];             /* procedure correlator identifier  */
   unsigned char   fqcp_name[17];       /* originator's network qualified   */
                                        /*  CP name                         */
   unsigned char   reserve3[3];         /* reserved                         */
} FQPCID
```

```
;
typedef struct session_stats
{
   AP_UINT16       rcv_ru_size;          /* session receive RU size          */
   AP_UINT16       send_ru_size;         /* session send RU size             */
   AP_UINT16       max_send_btu_size;    /* Maximum send BTU size            */
   AP_UINT16       max_rcv_btu_size;     /* Maximum rcv BTU size             */
   AP_UINT16       max_send_pac_win;     /* Maximum send pacing window size  */
   AP_UINT16       cur_send_pac_win;     /* Current send pacing window size  */
   AP_UINT16       max_rcv_pac_win;      /* Maximum receive pacing window    */
                                         /*  size                            */
   AP_UINT16       cur_rcv_pac_win;      /* Current receive pacing window    */
                                         /*  size                            */
   AP_UINT32       send_data_frames;     /* Number of data frames sent       */
   AP_UINT32       send_fmd_data_frames; /* Num fmd data frames sent         */
   AP_UINT32       send_data_bytes;      /* Number of data bytes sent        */
   AP_UINT32       rcv_data_frames;      /* Number of data frames received   */
   AP_UINT32       rcv_fmd_data_frames;  /* Num fmd data frames received     */
   AP_UINT32       rcv_data_bytes;       /* Number of data bytes received    */
   unsigned char   sidh;                 /* Session ID high byte (from LFSID)*/
   unsigned char   sidl;                 /* Session ID low byte (from LFSID) */
   unsigned char   odai;                 /* ODAI bit set                     */
   unsigned char   ls_name[8];           /* Link station name                */
   unsigned char   pacing_type;          /* type of pacing in use            */
} SESSION_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_SESSION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of sessions for which data should be returned. To request data for a specific session rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *session_id* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *session_id* parameter.

The combination of the local LU (*lu_name* or *lu_alias*), partner LU (*plu_alias* or *fqplu_name*), and *mode_name* specified is used as an index into the list of sessions if the *list_options* parameter is set to AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

lu_name

LU name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To specify that the LU is identified by its alias rather than its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter. To specify the LU associated with the local CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

lu_alias

Locally defined LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. This parameter is used only if *lu_name* is set to 8 binary zeros; it is ignored otherwise. To specify the LU associated with the local CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

plu_alias

Partner LU alias. To return information only about sessions associated with a specific partner LU, specify the partner LU alias (in this parameter) or the partner LU fully qualified name (in the following parameter). To return information about all sessions without filtering on the partner LU, set both of these parameters to binary zeros.

This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. To specify that the LU is identified by its LU name rather than its alias, set this parameter to 8 binary zeros and specify the LU name in the following parameter.

fqplu_name

Fully qualified network name for the partner LU. This parameter is used only if *plu_alias* is set to 8 binary zeros; it is ignored otherwise.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Mode name filter. To return information only about sessions associated with a specific mode, specify the mode name; the partner LU must also be specified (using one of the two preceding parameters). To return information about all sessions without filtering on mode name, set this parameter to 8 binary zeros.

The mode name is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

session_id

8-byte identifier of the session. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

session_summary.overlay_size

The size of the returned session_summary structure, and therefore the offset to the start of the next entry in the data buffer.

session_summary.plu_alias

Partner LU alias. This is an 8-byte ASCII character string, right-padded with ASCII spaces.

session_summary.fqplu_name

Fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

session_summary.mode_name

Mode name. This is an 8-byte type-A EBCDIC string (starting with a letter), right-padded with EBCDIC spaces.

session_summary.session_id

8-byte identifier of the session.

session_summary.fqpcid.pcid

Procedure Correlator ID. This is an 8-byte hexadecimal string.

session_summary.fqpcid.fqcp_name

Fully qualified CP name. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

session_detail.overlay_size

The size of the returned session_detail structure, and therefore the offset to the start of the next entry in the data buffer.

session_detail.plu_alias

Partner LU alias. This is an 8-byte ASCII character string, right-padded with ASCII spaces.

session_detail.fqplu_name

Fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

session_detail.mode_name

Mode name. This is an 8-byte type-A EBCDIC string (starting with a letter), right-padded with EBCDIC spaces.

session_detail.session_id

8-byte identifier of the session.

session_detail.fqpcid.pcid

Procedure Correlator ID. This is an 8-byte hexadecimal string.

session_detail.fqpcid.fqcp_name

Fully qualified control point name. This is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and a network name of 1-8 A-string characters.

session_detail.cos_name

Class of service name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

session_detail.trans_pri

Transmission priority. Possible values are:

AP_LOW

AP_MEDIUM

AP_HIGH

AP_NETWORK

session_detail.ltd_res

Specifies whether the session uses a limited resource link. Possible values are:

AP_YES

Session uses a limited resource link.

AP_NO

Session does not use a limited resource link.

session_detail.polarity

Specifies the polarity of the session. Possible values are:

AP_PRIMARY

AP_SECONDARY

session_detail.contention

Specifies whether the session is a contention winner or contention loser session for the local LU. Possible values are:

AP_CONWINNER

Contention winner session

AP_CONLOSER

Contention loser session

session_detail.sess_stats.rcv_ru_size

Maximum receive RU size.

session_detail.sess_stats.send_ru_size

Maximum send RU size.

session_detail.sess_stats.max_send_btu_size

Maximum BTU size that can be sent.

session_detail.sess_stats.max_rcv_btu_size

Maximum BTU size that can be received.

session_detail.sess_stats.max_send_pac_win

Maximum size of the send pacing window on this session.

session_detail.sess_stats.cur_send_pac_win

Current size of the send pacing window on this session.

session_detail.sess_stats.max_rcv_pac_win

Maximum size of the receive pacing window on this session.

session_detail.sess_stats.cur_rcv_pac_win

Current size of the receive pacing window on this session.

session_detail.sess_stats.send_data_frames

Number of normal flow data frames sent.

session_detail.sess_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

session_detail.sess_stats.send_data_bytes

Number of normal flow data bytes sent.

session_detail.sess_stats.rcv_data_frames

Number of normal flow data frames received.

session_detail.sess_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

session_detail.sess_stats.rcv_data_bytes

Number of normal flow data bytes received.

session_detail.sess_stats.sidh

Session ID high byte.

session_detail.sess_stats.sidl

Session ID low byte.

session_detail.sess_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station. It sets it to one if the BIND sender is the node containing the secondary link station.

session_detail.sess_stats.ls_name

Link station name associated with statistics. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. This field can be used to correlate the session statistics with the link over which session data flows.

session_detail.sess_stats.pacing_type

The type of receive pacing in use on this session. Possible values are:

```
AP_NONE
AP_FIXED
AP_ADAPTIVE
```

session_detail.duplex_support

Returns the conversation duplex support as negotiated on the BIND. Possible values are:

AP_HALF-DUPLEX

Only half-duplex conversations are supported.

AP_FULL_DUPLEX

Both full-duplex and half-duplex sessions are supported. Expedited data is also supported.

session_detail.sscp_id

For dependent LU sessions, this parameter is the SSCP ID received in the ACTPU from the host for the PU to which the local LU is mapped. For independent LU sessions, this parameter is set to 0 (zero).

session_detail.session_start_time

The time between the CP starting and this session becoming active, measured in one-hundredths of a

second. If the session is not fully active when the query is processed, this parameter is set to `0` (zero).

session_detail.session_timeout

The timeout associated with this session This timeout is derived from:

- The LU 6.2 timeout associated with the local LU
- The LU 6.2 timeout associated with the remote LU
- The mode timeout
- The global timeout
- The limited resource timeout (if this session is running over a limited resource link)

session_detail.plu_slu_comp_lvl

Specifies the compression level for data sent from the primary LU (PLU) to the secondary LU (SLU). Possible values are:

`AP_NONE`

Compression is not used.

`AP_RLE_COMPRESSION`

Run-length encoding (RLE) compression is used.

session_detail.slu_plu_comp_lvl

Specifies the compression level for data sent from the secondary LU (SLU) to the primary LU (PLU). Possible values are:

`AP_NONE`

Compression is not used.

`AP_RLE_COMPRESSION`

Run-length encoding (RLE) compression is used.

session_detail.rscv_len

Length of the RSCV which is appended to the `session_detail` structure. (If none is appended, then the length is zero.)

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LU_ALIAS

The specified *lu_alias* parameter was not valid.

AP_INVALID_LU_NAME

The specified *lu_name* parameter was not valid.

AP_INVALID_SESSION_ID

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied value, but the *session_id* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_SNA_NET

QUERY_SNA_NET returns information about servers that can act as backup master servers, as defined in the sna.net file. It can be used to obtain information about a specific server or about multiple servers, depending on the options used.

The ordering of server names in this file is significant; the first server listed in the file will always be the master if it is active, the second will be the master if the first is inactive, the third will be the master if the first and second are both inactive, and so on. Because of this, the list of server names returned on QUERY_SNA_NET is in the same order as it is in the file; the returned names are not ordered by name length and lexicographical ordering, as with other QUERY_* verbs.

This verb must be issued to the sna.net file.

VCB Structure

```
typedef struct query_sna_net
{
   AP_UINT16        opcode;                /* Verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* Primary return code          */
   AP_UINT32        secondary_rc;          /* Secondary return code        */
   unsigned char    *buf_ptr;              /* pointer to buffer            */
   AP_UINT32        buf_size;              /* buffer size                  */
   AP_UINT32        total_buf_size;        /* total buffer size required   */
   AP_UINT16        num_entries;           /* number of entries            */
   AP_UINT16        total_num_entries;     /* total number of entries      */
   unsigned char    list_options;          /* listing options              */
   unsigned char    reserv3;               /* reserved                     */
   unsigned char    security;              /* security for Windows clients */
   unsigned char    domain_name[64];       /* domain name                  */
   unsigned char    server_name[64];       /* master or backup server name */
   unsigned char    reserv4[4];            /* reserved                     */
} QUERY_SNA_NET;


typedef struct backup_summary
{
   AP_UINT16        overlay_size;          /* size of returned entry       */
   unsigned char    reserv1[2];            /* reserved                     */
```

```
   unsigned char    server_name[64];         /* master or backup server name */
   unsigned char    reserv2[4];              /* reserved                     */
} BACKUP_SUMMARY;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_SNA_NET

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of server names for which data should be returned. To request a specific entry rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data.

Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *server_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *server_name* parameter.

For more information about how the application can obtain specific entries from the list, see "List Options For QUERY_* Verbs". The server names are listed in the same order as in the file, not in order of name length and/or lexicographical order as for other QUERY_* verbs.

server_name

Name of the server for which information is required, or the name to be used as an index into the list of servers. The server name is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than

num_entries indicates that not all the available entries were returned.

security

The level of security for Windows clients accessing SNAplus2 servers. This is specified during installation, or can be modified later using the snapwinsec program; for more information about security for Windows clients, refer to the *HP-UX SNAplus2 Administration Guide*. Possible values are:

AP_SECURITY_OFF

No security restrictions.

AP_SECURITY_DOMAIN

Login IDs and passwords used by Windows Client users must be defined on the HP-UX system.

domain_name

The name of the TCP/IP domain containing the SNAplus2 LAN. This name was specified during installation of the master server.

Each entry in the data buffer consists of the following parameters:

backup_summary.overlay_size

The size of the returned backup_summary structure, and therefore the offset to the start of the next entry in the data buffer.

backup_summary.server_name

Server name.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters.

primary_rc

AP_STATE_CHECK

secondary_rc

Possible values are:

AP_RECORD_NOT_FOUND

The *list_options* parameter was set to AP_LIST_INCLUSIVE or AP_LIST_FROM_NEXT to list entries starting from the supplied server name, but the *backup_name* parameter did not match an entry in the file. If the supplied name was one returned on a previous QUERY_SNA_NET verb, this indicates that the list has been updated (by another administration program or NOF application) since the previous verb; the application should reissue QUERY_SNA_NET to obtain the complete list.

AP_INVALID_TARGET

The target handle on the NOF API call specified a configuration file or a node. This verb must be issued to the sna.net file.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_STATISTICS

QUERY_STATISTICS returns statistics on the usage of an LS or port. The type of information returned depends on the DLC type. The QLLC software does not support link statistics; do not issue this verb for a QLLC port or LS.

For SDLC, the verb returns either statistics (counts of events such as particular frame types sent or received) or operational information (details of parameters currently being used), for either an LS or a port.

For Token Ring, Ethernet, or FDDI, the verb returns statistics information for an LS.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_statistics
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* primary return code          */
   AP_UINT32        secondary_rc;          /* secondary return code        */
   unsigned char    name[8];               /* LS name or port name         */
   unsigned char    stats_type;            /* LS or port statistics?       */
   unsigned char    table_type;            /* statistics table requested   */
   unsigned char    reset_stats;           /* reset the statistics?        */
   unsigned char    dlc_type;              /* type of DLC                  */
   unsigned char    statistics[256];       /* current statistics           */
   unsigned char    reserva[20];           /* reserved                     */
} QUERY_STATISTICS;
```

LS statistics for SDLC:

```
typedef struct sdl_ls_stats_table
{
  V0_MUX_INFO    mux_info;          /* streams config info                  */
  AP_UINT32      index;             /* index of port that owns LS           */
  unsigned int   address;           /* poll address of secondary link station*/
  unsigned char  reserv[3];         /* reserved                             */
  AP_UINT32      blus_in;           /* frames received from adjacent link   */
                                    /*  station                             */
```

```
  AP_UINT32      blus_out;          /* frames sent to adjacent link station  */
  AP_UINT32      octets_in;         /* bytes received from adjacent link     */
                                    /*  station                              */
  AP_UINT32      octets_out;        /* bytes sent to adjacent link station   */
  AP_UINT32      polls_out;         /* polls sent to adjacent link station   */
  AP_UINT32      poll_rsps_out;     /* polls responded to by adjacent link   */
                                    /*  station                              */
  AP_UINT32      local_busies;      /* number of times local link station has*/
                                    /*  entered busy state (RNR)             */
  AP_UINT32      remote_busies;     /* number of times remote link station   */
                                    /*  has entered busy state (RNR)         */
  AP_UINT32      iframes_in;        /* I-frames rcvd from adjacent link      */
                                    /*  station                              */
  AP_UINT32      iframes_out;       /* I-frames sent to adjacent link station*/
  AP_UINT32      retransmits_in;    /* Total number of retransmitted         */
                                    /* I-frames received                     */
  AP_UINT32      retransmits_out;   /* I-frames retransmitted since LS       */
                                    /*  start-up                             */
  AP_UINT32      ioctets_in;        /* bytes in I-frames received            */
  AP_UINT32      ioctets_out;       /* bytes in I-frames sent                */
  AP_UINT32      uiframes_in;       /* reserved                              */
  AP_UINT32      uiframes_out;      /* reserved                              */
  AP_UINT32      xids_in;           /* XIDs rcvd from adjacent link station  */
  AP_UINT32      xids_out;          /* XIDs sent to adjacent link station    */
  AP_UINT32      tests_in;          /* TEST frames received                  */
  AP_UINT32      tests_out;         /* TEST frames sent                      */
  AP_UINT32      rejs_in;           /* REJ frames received                   */
  AP_UINT32      rejs_out;          /* REJ frames sent                       */
  AP_UINT32      frmrs_in;          /* FRMR frames received                  */
  AP_UINT32      frmrs_out;         /* FRMR frames sent                      */
  AP_UINT32      sims_in;           /* SIM frames received                   */
  AP_UINT32      sims_out;          /* SIM frames sent                       */
  AP_UINT32      rims_in;           /* RIM frames received                   */
  AP_UINT32      rims_out;          /* RIM frames sent                       */
  AP_UINT32      disc_in;           /* reserved                              */
  AP_UINT32      disc_out;          /* reserved                              */
  AP_UINT32      ua_in;             /* reserved                              */
  AP_UINT32      ua_out;            /* reserved                              */
  AP_UINT32      dm_in;             /* reserved                              */
  AP_UINT32      dm_out;            /* reserved                              */
  AP_UINT32      snrm_in;           /* SNRM frames received                  */
  AP_UINT32      snrm_out;          /* SNRM frames sent                      */
} SDL_LS_STATS_TABLE;
```

LS operational information for SDLC:

```
typedef struct sdl_ls_oper_table
{
  V0_MUX_INFO    mux_info;       /* streams config info                      */
  AP_UINT32      index;          /* index of port that owns LS               */
  unsigned char  address;        /* poll address of secondary link station   */
  unsigned char  reserve;        /* reserved                                 */
  AP_UINT16      role;           /* current role of link station             */
  unsigned char  name[8];        /* reserved                                 */
  AP_UINT16      state;          /* operational state of LS                  */
  AP_UINT16      maxdata;        /* current max PDU size for logical link    */
  AP_UINT32      replyto;        /* current reply timeout                    */
  AP_UINT32      maxin;          /* current max unack'd frames LS can receive*/
  AP_UINT32      maxout;         /* current max unack'd frames LS can send   */
  unsigned char  modulo;         /* sequence number modulus                  */
  unsigned char  reserv2[3];     /* reserved                                 */
  AP_UINT32      retries_m;      /* number of retries in a retry sequence    */
  AP_UINT32      retries_t;      /* interval between retry sequences         */
  AP_UINT32      retries_n;      /* number of times to repeat retry sequence */
  AP_UINT32      rnrlimit;       /* how long adjacent LS can be in RNR state */
                                 /*  before it is considered inoperative     */
  unsigned char  datmode;        /* communications mode with adjacent LS     */
  unsigned char  last_fail_cause;         /* reserved                        */
  unsigned char  last_fail_ctrl_in[2];    /* control field of last frame rcvd*/
                                          /*  before last failure            */
  unsigned char  last_fail_ctrl_out[2];   /* control field of last frame sent*/
                                          /*  before last failure            */
  unsigned char  last_fail_frmr_info[5];  /* info field of FRMR frame if     */
                                          /*  last failure was caused by     */
                                          /*  invalid frame                  */
  unsigned char  sdoppad1;                /* reserved                        */
  AP_UINT32      last_fail_replyto_s;     /* number of REPLYTO timeouts at   */
                                          /*  time of last failure           */
  unsigned char  g_poll;         /* group poll address                       */
  unsigned char  sim_rim;        /* are SIM / RIM supported?                 */
  unsigned char  xmit_rcv_cap;   /* transmit / receive capability            */
} SDL_LS_OPER_TABLE;
```

Port statistics for SDLC:

```
typedef struct sdl_port_stats_table
{
  V0_MUX_INFO    mux_info;        /* streams config info                     */
  AP_UINT32      index;           /* index of port                           */
  AP_UINT32      dwarf_frames;    /* frames received too short to be valid   */
  AP_UINT32      polls_out;       /* polls sent to adjacent link stations    */
  AP_UINT32      poll_rsps_out;   /* polls responded to by adjacent link stns*/
```

```
  AP_UINT32      local_busies;    /* number of times local link station      */
                                  /*  has entered busy state (RNR)           */
  AP_UINT32      remote_busies;   /* number of times remote link stations    */
                                  /*  have entered busy state (RNR)          */
  AP_UINT32      iframes_in;      /* I-frames rcvd from adjacent link        */
                                  /*  stations                               */
  AP_UINT32      iframes_out;     /* I-frames sent to adjacent link stations */
  AP_UINT32      octets_in;       /* bytes received from adjacent link       */
                                  /*  stations                               */
  AP_UINT32      octets_out;      /* bytes sent to adjacent link stations    */
  AP_UINT32      protocol_errs;   /* link deactivations due to bad rcvd      */
                                  /*  frames                                 */
  AP_UINT32      activity_to_s;   /* link deactivations due to inactivity    */
  AP_UINT32      rnrlimit_s;      /* link deacts due to rem busy timer expiry*/
  AP_UINT32      retries_exps;    /* link deacts due to end of retry sequence*/
  AP_UINT32      retransmits_in;  /* retransmitted I-frames rcvd since       */
                                  /*  start-up                               */
  AP_UINT32      retransmits_out; /* I-frames retransmitted  since start-up  */
} SDL_PORT_STATS_TABLE;
```

Port operational information for SDLC:

```
typedef struct sdl_port_oper_table
{
  V0_MUX_INFO     mux_info;           /* streams config info               */
  AP_UINT32       index;              /* index of port                     */
  unsigned char   name[8];            /* reserved                          */
  unsigned char   role;               /* current role of link station(s)   */
                                      /*  using port                       */
  unsigned char   type;               /* line type - leased or switched    */
  unsigned char   topology;           /* can port be point-to-point or     */
                                      /*  multipoint                       */
  unsigned char   reserve;            /* reserved                          */
  AP_UINT32       activto;            /* how long switched line can be     */
                                      /*  inactive before port disconnects */
  AP_UINT32       pause;              /* time between poll cycles          */
  unsigned char   slow_poll_method;   /* slow poll method                  */
  unsigned char   reserv2[3];         /* reserved                          */
  AP_UINT32       slow_poll_timer;    /* slow poll timer                   */
  unsigned char   last_fail_cause;    /* reserved                          */
} SDL_PORT_OPER_TABLE;
```

LS statistics for Token Ring, Ethernet, FDDI:

```
typedef struct vdl_ls_statistics
{
```

```
  V0_MUX_INFO    mux_info;             /* streams config info               */
  AP_UINT32      ls_st_mus_sent;       /* Frames sent from this Link Station */
  AP_UINT32      ls_st_mus_received;   /* Frames received at this Link       */
                                       /*  Station                          */
  AP_UINT32      ls_st_bytes_sent;     /* Bytes sent from this Link Station  */
  AP_UINT32      ls_st_bytes_received; /* Bytes received at this Link Station*/
} VDL_LS_STATISTICS;

typedef struct v0_mux_info
{
   AP_UINT16        dlc_type;             /* DLC implementation type        */
   unsigned char    need_vrfy_fixup;      /* reserved                       */
   unsigned char    num_mux_ids;          /* reserved                       */
   AP_UINT32        card_type;            /* type of adapter card           */
   AP_UINT32        adapter_number;       /* DLC adapter number             */
   AP_UINT32        oem_data_length;      /* reserved                       */
   int              mux_ids[5];           /* reserved                       */
} V0_MUX_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_STATISTICS

name

Name of the LS or port for which statistics are required (as specified by the *stats_type* parameter). This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. SNAplus2 uses this name to correlate the response to the correct link station or port.

stats_type

The type of resource for which statistics are requested.

Allowed values for SDLC:

AP_LS

Return LS statistics.

AP_PORT

Return port statistics.

For Token Ring / Ethernet / FDDI, this must be set to AP_LS.

table_type

The type of statistics information requested.

Allowed values for SDLC:

AP_STATS_TBL

Statistical information.

AP_OPER_TBL

Operational information.

For Token Ring / Ethernet / FDDI , this must be set to AP_STATS_TBL.

reset_stats

Specifies whether to reset the statistics when this verb completes. This parameter applies only if *table_type* is set to AP_STATS_TBL; it is ignored otherwise. Possible values are:

AP_YES

Reset the statistics; a subsequent QUERY_STATISTICS verb will contain only data gathered after this verb was issued.

AP_NO

Do not reset the statistics; the data on this verb will be included in the data returned to a subsequent QUERY_STATISTICS verb.

dlc_type

Type of the DLC. Possible values are:

AP_SDLC

Synchronous data link control

AP_TR

Token Ring

AP_ETHERNET

Ethernet

AP_FDDI

Fiber distributed data interface

AP_X25

X.25 packet switching

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

dlc_type

Type of DLC for which statistics information is being returned. Possible values are:

AP_SDLC

SDLC

AP_X25

QLLC

AP_TR

Token Ring

AP_ETHERNET

Ethernet

AP_FDDI

FDDI

AP_X25

X.25 packet switching

AP_LLC2

LLC2

AP_DLSW

Data link switching

AP_TWINAX

Twinaxial cable

AP_ISDN

Integrated services digital network

AP_ATM

Asynchronous transfer mode

AP_ASYNC

Asynchronous

statistics

Current statistics for the link station or port. This string is replaced by the appropriate structure for the DLC type. The parameters in the structure are described below.

dlc_type, *card_type*, *adapter_number*

Streams configuration information for the DLC. For more information about these parameters, see "DEFINE_DLC".

LS statistics for SDLC:

sdl_ls_stats_table.index

The index value used internally by SNAplus2 to identify the port that owns this LS.

sdl_ls_stats_table.address

The poll address of the secondary link station.

sdl_ls_stats_table.blus_in

The total number of basic link units (frames) received from the adjacent link station.

sdl_ls_stats_table.blus_out

The total number of basic link units (frames) transmitted to the adjacent link station.

sdl_ls_stats_table.octets_in

The total number of bytes (not including FCSs) received from the adjacent link station.

sdl_ls_stats_table.octets_out

The total number of bytes (not including FCSs) transmitted to the adjacent link station.

sdl_ls_stats_table.polls_out

Total number of polls sent to the adjacent link station.

sdl_ls_stats_table.poll_rsps_out

Total number of polls responded to by the adjacent link station.

sdl_ls_stats_table.local_busies

Total number of times the local link station has entered busy state (RNR).

sdl_ls_stats_table.remote_busies

Total number of times the remote link station has entered busy state (RNR).

sdl_ls_stats_table.iframes_in

The total number of I-frames received from the adjacent link station (including retries and out-of-order frames).

sdl_ls_stats_table.iframes_out

The total number of I-frames transmitted to the adjacent link station (including retries and out-of-order frames).

sdl_ls_stats_table.retransmits_in

The total number of retransmissions of I-frames received.

sdl_ls_stats_table.retransmits_out

The total number of retransmissions of I-frames to the adjacent link station.

sdl_ls_stats_table.ioctets_in

The total number of bytes in I-frames received from the adjacent link station.

sdl_ls_stats_table.ioctets_out

The total number of bytes in I-frames transmitted to the adjacent link station.

sdl_ls_stats_table.xids_in

The total number of XID frames received from the adjacent link station.

sdl_ls_stats_table.xids_out

The total number of XID frames transmitted to the adjacent link station.

sdl_ls_stats_table.tests_in

The total number of TEST frames, commands, or responses received from the adjacent link station.

sdl_ls_stats_table.tests_out

The total number of TEST frames, commands, or responses transmitted to the adjacent link station.

sdl_ls_stats_table.rejs_in

The total number of REJ frames received from the adjacent link station.

sdl_ls_stats_table.rejs_out

The total number of REJ frames transmitted to the adjacent link station.

sdl_ls_stats_table.frmrs_in

The total number of Frame Reject frames received from the adjacent link station.

sdl_ls_stats_table.frmrs_out

The total number of Frame Reject frames transmitted to the adjacent link station.

sdl_ls_stats_table.sims_in

The total number of Set Initialization Mode frames received from the adjacent link station.

sdl_ls_stats_table.sims_out

The total number of Set Initialization Mode frames transmitted to the adjacent link station.

sdl_ls_stats_table.rims_in

The total number of Request Initialization Mode frames received from the adjacent link station.

sdl_ls_stats_table.rims_out

The total number of Request Initialization Mode frames transmitted to the adjacent link station.

sdl_ls_stats_table.snrm_in

The total number of SNRM frames received.

sdl_ls_stats_table.snrm_out

The total number of SNRM frames sent.

LS operational information for SDLC:

sdl_ls_oper_table.index

The index value used internally by SNAplus2 to identify the port that owns this LS.

sdl_ls_stats_table.address

The poll address of the secondary link station.

sdl_ls_stats_table.role

The link role of the LS. Possible values are:

SDL_MIB_PRIMARY

Primary

SDL_MIB_SECONDARY

Secondary

SDL_MIB_NEGOTIABLE

Negotiable

sdl_ls_stats_table.state

An internal value indicating the processing state of the LS software (for use by support personnel).

sdl_ls_stats_table.maxdata

The current maximum PDU size allowed for the logical link (the size includes the TH and RH). For a switched line, this value may be negotiated during XID exchange.

sdl_ls_stats_table.replyto

The current reply timeout, in hundredths of a second. This parameter applies only if the LS role is primary; its value is undefined if the LS role is secondary.

sdl_ls_stats_table.maxin

The maximum number of frames that the LS can receive before it must send an acknowledgment.

sdl_ls_stats_table.maxout

The maximum number of frames that the LS can send before it must wait for an acknowledgment.

sdl_ls_stats_table.modulo

The sequence number modulus for the LS. Possible values are:

SDL_MIB_EIGHT

8

SDL_MIB_ONETWENTYEIGHT

128

sdl_ls_stats_table.retries_m

The maximum number of frames in a retry sequence (a sequence of frames that the LS retransmits because it has not received a positive acknowledgment for them).

sdl_ls_stats_table.retries_t

The timeout between retransmissions of a retry sequence.

sdl_ls_stats_table.retries_n

The number of times that the LS attempts to retransmit a retry sequence.

sdl_ls_stats_table.rnrlimit

The maximum length of time that the adjacent LS can remain in RNR state before the local LS considers it to be inoperative.

sdl_ls_stats_table.datmode

The communications mode with the adjacent LS. Possible values are:

SDL_MIB_HALF

Two-way alternate (half-duplex)

SDL_MIB_FULL

Two-way simultaneous (full-duplex)

sdl_ls_stats_table.last_fail_ctrl_in

The control field from the last frame received before the last failure. If the LS has not failed, this field is set to zeros.

sdl_ls_stats_table.last_fail_ctrl_out

The control field from the last frame sent before the last failure. If the LS has not failed, this field is set to zeros.

sdl_ls_stats_table.last_fail_frmr_info

If the last failure was caused by a frame that was not valid, this parameter contains the information field from the FRMR frame. If the LS has not failed, or if the failure cause was not a frame that was not valid, this field is set to zeros.

sdl_ls_stats_table.last_fail_replyto_s

The number of times that the reply timeout expired before the last failure. If the LS has not failed, this field is set to zero.

sdl_ls_stats_table.g_poll

The group poll address for the LS. If the LS is not in a group, this field is set to zero.

sdl_ls_stats_table.sim_rim

Specifies whether the LS supports transmission of SIM and RIM control frames. Possible values are:

SDL_MIB_YES

LS supports SIM and RIM.

SDL_MIB_NOLS

does not support SIM and RIM.

sdl_ls_stats_table.xmit_rcv_cap

Specifies the LS's transmit / receive capability. Possible values are:

SDL_MIB_HALF

Half-duplex

SDL_MIB_FULL

Full-duplex

Port statistics for SDLC:

sdl_port_stats_table.index

The index value used internally by SNAplus2 to identify the port.

sdl_port_stats_table.dwarf_frames

The number of frames received by the port that were too short to be valid.

sdl_port_stats_table.polls_out

Total number of polls sent to adjacent link stations.

sdl_port_stats_table.poll_rsps_out

Total number of polls responded to by adjacent link stations.

sdl_port_stats_table.local_busies

Total number of times the local link station has entered busy state (RNR).

sdl_port_stats_table.remote_busies

Total number of times remote link stations have entered busy state (RNR).

sdl_port_stats_table.iframes_in

The total number of I-frames received from adjacent link stations (including retries and out-of-order frames).

sdl_port_stats_table.iframes_out

The total number of I-frames transmitted to adjacent link stations (including retries and out-of-order frames).

sdl_port_stats_table.octets_in

The total number of bytes (not including FCSs) received from adjacent link stations.

sdl_port_stats_table.octets_out

The total number of bytes (not including FCSs) transmitted to adjacent link stations.

sdl_port_stats_table.protocol_errs

The number of times that SNAplus2 has deactivated an LS using this port because a frame received from the adjacent link station contained a protocol error.

sdl_port_stats_table.activity_to_s

The number of times that SNAplus2 has deactivated an LS using this port because there was no activity on the link.

sdl_port_stats_table.rnrlimit_s

The number of times that SNAplus2 has deactivated an LS using this port because the Remote Busy timer expired.

sdl_port_stats_table.retries_exps

The number of times that SNAplus2 has deactivated an LS using this port because a retry sequence has been exhausted.

sdl_port_stats_table.retransmits_in

The total number of retransmitted I-frames received from adjacent link stations.

sdl_port_stats_table.retransmits_out

The total number of retransmissions of I-frames to adjacent link stations.

Port operational information for SDLC:

sdl_port_oper_table.index

The index value used internally by SNAplus2 to identify the port.

sdl_port_oper_table.role

The link role of the port. Possible values are:

SDL_MIB_PRIMARY

Primary

SDL_MIB_SECONDARY

Secondary

SDL_MIB_NEGOTIABLE

Negotiable

sdl_port_oper_table.type

Specifies whether the port is operating as though connected to a leased or switched line. Possible values are:

SDL_MIB_LEASED

SDL_MIB_SWITCHED

sdl_port_oper_table.topology

Specifies whether the port can operate in a multipoint topology. Possible values are:

SDL_MIB_POINT_TO_POINT

Port can operate only as point-to-point.

SDL_MIB_MULTIPOINT

Port can operate as multipoint.

sdl_port_oper_table.activto

The length of time, in hundredths of a second, that the

port allows a switched line to remain inactive (no I-frames being transferred) before disconnecting. A value of zero indicates no timeout; the line remains connected regardless of inactivity. This parameter applies only for a switched link; its value is undefined for a leased link.

sdl_port_oper_table.pause

The length of time that the primary station waits between successive cycles of polling secondary stations. This parameter applies only if the LS role is primary; its value is undefined if the LS role is secondary.

sdl_port_oper_table.slow_poll_method

The method used for periodically polling failed secondary link stations. This is set to SDL_MIB_POLLPAUSE.

sdl_port_oper_table.slow_poll_timer

The timeout between polls for failed secondary link stations. This parameter applies only if the port is primary and operating in a multipoint topology; its value is undefined otherwise.

LS statistics for Token Ring, Ethernet , FDDI:

vdl_ls_statistics.ls_st_mus_sent

Number of frames sent from SNAplus2 on this LS since the LS was started.

vdl_ls_statistics.ls_st_mus_received

Number of frames received by SNAplus2 on this LS since the LS was started.

vdl_ls_statistics.ls_st_bytes_sent

Number of bytes sent from SNAplus2 on this LS since the LS was started.

vdl_ls_statistics.ls_st_bytes_received

Number of bytes received by SNAplus2 on this LS since the LS was started.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LINK_NAME

The supplied name parameter was not a valid LS name.

AP_INVALID_PORT_NAME

The supplied name parameter was not a valid port name.

AP_INVALID_STATS_TYPE

The *stats_type* parameter was not set to a valid value.

AP_INVALID_TABLE_TYPE

The *table_type* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc

AP_STATE_CHECK

secondary_rc

Possible values are:

AP_LINK_DEACTIVATED

The specified link is not currently active.

AP_PORT_DEACTIVATED

The specified port is not currently active.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute because the DLC type does not support returning statistics information, SNAplus2 returns the following parameter:

primary_rc

AP_FUNCTION_NOT_SUPPORTED

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TN_SERVER_TRACE

This verb returns information about the current tracing options for the SNAplus2 TN server feature.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_tn_server_trace
{
   AP_UINT16        opcode;                    /* verb operation code     */
   unsigned char    reserv2;                   /* reserved                */
   unsigned char    format;                    /* reserved                */
   AP_UINT16        primary_rc;                /* primary return code     */
   AP_UINT32        secondary_rc;              /* secondary return code   */
   AP_UINT16        trace_flags;               /* trace flags             */
} QUERY_TN_SERVER_TRACE;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_TN_SERVER_TRACE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

trace_flags

The types of tracing currently active.

If no tracing is active, or if tracing of all types is active,

this is one of the following values:

AP_TN_SERVER_NO_TRACE

No tracing.

AP_TN_SERVER_ALL_TRACE

Tracing of all types.

If tracing is being used on specific interfaces, this parameter is set to one or more values from the list below, combined using a logical OR operation.

AP_TN_SERVER_TRC_TCP

TCP/IP interface tracing: messages between TN server and TN3270 clients

AP_TN_SERVER_TRC_FM

Node interface tracing: internal control messages, and messages between TN server and TN3270 clients (in internal format)

AP_TN_SERVER_TRC_CFG

Configuration message tracing: messages relating to the configuration of TN server

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TN3270_ACCESS_DEF

QUERY_TN3270_ACCESS_DEF returns information about TN3270 users on other computers that can use the TN server feature of SNAplus2 to access a host for 3270 emulation. It can return either summary or detailed information, about a single user or multiple users, depending on the options used.

VCB Structure

```
typedef struct query_tn3270_access_def
{
   AP_UINT16        opcode;                /* verb operation code           */
   unsigned char    reserv2;               /* reserved                      */
   unsigned char    format;                /* reserved                      */
   AP_UINT16        primary_rc;            /* primary return code           */
   AP_UINT32        secondary_rc;          /* secondary return code         */
   unsigned char    *buf_ptr;              /* pointer to buffer             */
   AP_UINT32        buf_size;              /* buffer size                   */
   AP_UINT32        total_buf_size;        /* total buffer size required    */
   AP_UINT16        num_entries;           /* number of entries             */
   AP_UINT16        total_num_entries;     /* total number of entries       */
   unsigned char    list_options;          /* listing options               */
   unsigned char    reserv3;               /* reserved                      */
   AP_UINT16        default_record;        /* start with DEFAULT record?    */
   unsigned char    client_address[68];    /* address of TN3270 user        */
   AP_UINT16        port_number;           /* TCP/IP port to access server  */
   AP_UINT32        num_init_sessions;     /* number of sessions for first  */
                                           /* user when starting in middle  */
   AP_UINT32        num_last_sessions;     /* number of sessions on last    */
                                           /*  detail overlay if last user  */
                                           /*  is incomplete                */
   unsigned char    last_user_incomplete;  /* set to AP_YES if session      */
                                           /*  data for last user incomplete*/
} QUERY_TN3270_ACCESS_DEF;

typedef struct tn3270_access_summary
{
   AP_UINT16        default_record;        /* is this the DEFAULT record?   */
   unsigned char    client_address[68];    /* address of TN3270 user        */
   AP_UINT16        address_format;        /* Format of client address      */
} TN3270_ACCESS_SUMMARY;
```

```
typedef struct tn3270_access_detail
{
  AP_UINT16              default_record;       /* is this the DEFAULT record?*/
  unsigned char          client_address[68];   /* address of TN3270 user     */
  AP_UINT32              num_filtered_sessions;/* num sess returned for user */
  TN3270_ACCESS_DEF_DATA def_data;             /* user definition            */
} TN3270_ACCESS_DETAIL;


typedef struct tn3270_access_def_data
{
   unsigned char   description[32];     /* Description - null terminated     */
   unsigned char   reserv1[16];         /* reserved                          */
   AP_UINT16       address_format;      /* Format of client address          */
   AP_UINT32       num_sessions;        /* Number of sessions being added    */
} TN3270_ACCESS_DEF_DATA;
```

For each session, up to the number specified by the *num_sessions* parameter, the following structure is included at the end of the def_data structure:

```
typedef struct tn3270_session_def_data
{
   AP_UINT16           sub_overlay_size;   /* reserved                      */
   unsigned char       description[32];    /* Session description           */
   unsigned char       tn3270_support;     /* Level of TN3270 support       */
   unsigned char       allow_specific_lu;  /* Allow access to specific LUs  */
   unsigned char       printer_lu_name[8]; /* Generic printer LU/pool       */
                                           /*   accessed                    */
   unsigned char       reserv1[6];         /* reserved                      */
   AP_UINT16           port_number;        /* TCP/IP port used to access    */
                                           /*  server                       */
   unsigned char       lu_name[8];         /* Generic display LU/pool       */
                                           /*  accessed                     */
   unsigned char       session_type;       /* Unused in SNAplus2 V6         */
   unsigned char       model_override;     /* Unused in SNAplus2 V6         */
   unsigned char       reserv3[4];         /* reserved                      */
   AP_UINT32           reserv4;            /* reserved                      */
} TN3270_SESSION_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_TN3270_ACCESS_DEF

sub_overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of users for which data should be returned. If detailed information about user sessions is being returned, this number includes partial entries (for which a client address is specified, so that the returned data does not include the user definition or the user's first session).

To request data for a specific user rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical `OR` operation with

one of the following values:

AP_FIRST_IN_LIST

Start at the first session for the first user in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the supplied client address and port number, or start at the first session for the specified client address if no port number is specified.

AP_LIST_FROM_NEXT

If a port number is specified, start at the session immediately following the session with the specified port number. If no port number is specified, start at the first session for the specified client address.

The list is ordered by client address and then by port number for each user. For more information about how the list is ordered and how the application can obtain specific entries from it, see "List Options For QUERY_* Verbs".

default_record

Specifies whether the requested entry (or the entry to be used as an index into the list) is the default record.

To query the default record, which is used by any TN3270 user not explicitly identified by a TCP/IP address, specify AP_YES. In this case, the *client_address* parameter is reserved.

To query a normal TN3270 user record, specify AP_NO.

client_address

The TCP/IP address of the TN3270 user for whom information is required, or the name to be used as an index into the list of users. This parameter is a string of 1-64 characters, followed by a null character; it is ignored if *list_options* is set to AP_FIRST_IN_LIST.

The address can be specified as a dotted-decimal IP address (such as 193.1.11.100), as a fully qualified name (such as newbox.this.co.uk), or as an alias

(such as newbox).

port_number

To return information starting with a specific session for the specified user, set this parameter to the TCP/IP port number defined for that session. To return information starting at the first session for the specified user, set this parameter to zero.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than *buf_size*.

total_num_entries

Total number of entries that could have been returned. This may be higher than *num_entries*.

num_entries

The number of entries actually returned. The last entry may be incomplete; this is indicated by the *last_user_incomplete* parameter.

num_init_sessions

If the *port_number* parameter was set to a nonzero value, so that the information for the first user in the list does not start with the user's first session, this parameter indicates the number of session structures for this user that are included in the returned data. Otherwise, this parameter is not used.

num_last_sessions

If the *last_user_incomplete* parameter indicates that the data for the last user is incomplete, this parameter indicates the number of session structures for this user that are included in the returned data. Otherwise, this parameter is not used.

last_user_incomplete

Specifies whether the information for the last user is incomplete. Possible values are:

AP_YES

The complete data for the last user was too large to fit in the data buffer. At least one session structure is included, but there are further session structures that are not included in the data buffer. The *num_last_sessions* parameter indicates how many session structures have been returned; the application can issue further verbs to obtain the remaining data.

AP_NO

The data for the last user is complete.

Each entry in the data buffer consists of the following:

tn3270_access_summary.default_record

Specifies whether this entry is the default record. Possible values are:

AP_YES

This is the default record. The *client_address* parameter is reserved.

AP_NO

This is a normal TN3270 user record.

tn3270_access_summary.client_address

The TCP/IP address of the TN3270 user. This is a null-terminated ASCII string.

tn3270_access_summary.address_format

Specifies the format of the *client_address* parameter. Possible values are:

AP_ADDRESS_IP

IP address

AP_ADDRESS_FQN

Alias or fully qualified name

tn3270_access_detail.default_record

Specifies whether this entry is the default record. Possible values are:

AP_YES

This is the default record. The *client_address* parameter is reserved.

AP_NO

This is a normal TN3270 user record.

tn3270_access_detail.client_address

The TCP/IP address of the TN3270 user. This is a null-terminated ASCII string.

tn3270_access_detail.num_filtered_sessions

The number of sessions returned for this user.

tn3270_access_detail.def_data

The details of the user, as defined in the configuration. This is followed by a number of session structures defining the user's sessions. The format of this information is the same as for the DEFINE_TN3270_ACCESS verb, except for the following:

- The *num_sessions* parameter in the def_data structure defines the total number of sessions defined for the user.
- If the *port_number* parameter was set to a nonzero value, the data for the first user will contain only the remaining session structures (starting from the requested entry), without the def_data structure.
- If the *last_user_incomplete* parameter is set to AP_YES, the total number of session structures

returned for the last user will be as specified by the *num_last_sessions* parameter; this will be less than *num_sessions*.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_CLIENT_ADDRESS

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *client_address* parameter did not match the address of any defined TN3270 user.

AP_INVALID_PORT_NUMBER

The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the *port_number* parameter did not match a port number defined for the specified TN3270 user.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TN3270_ASSOCIATION

QUERY_TN3270_ASSOCIATION returns information about associations between display LUs and printer LUs. Associations are queried by display LU name and are returned in order of display LU name.

This verb can be used to obtain information about a specific association or about multiple associations, depending on the options used.

VCB Structure

```
typedef struct query_tn3270_association
{
   AP_UINT16        opcode;               /* verb operation code           */
   unsigned char    reserv2;              /* reserved                      */
   unsigned char    format;               /* reserved                      */
   AP_UINT16        primary_rc;           /* primary return code           */
   AP_UINT32        secondary_rc;         /* secondary return code         */
   unsigned char   *buf_ptr;              /* pointer to buffer             */
   AP_UINT32        buf_size;             /* buffer size                   */
   AP_UINT32        total_buf_size;       /* total buffer size required    */
   AP_UINT16        num_entries;          /* number of entries             */
   AP_UINT16        total_num_entries;    /* total number of entries       */
   unsigned char    list_options;         /* listing options               */
   unsigned char    reserv3;              /* reserved                      */
   unsigned char    display_lu_name[8];   /* Display LU name               */
} QUERY_TN3270_ASSOCIATION;


typedef struct tn3270_association
{
   AP_UINT16      overlay_size;           /* Overlay size                  */
   unsigned char  reserv2[2];             /* reserved                      */
   unsigned char  display_lu_name[8];     /* Display LU name               */
   TN3270_ASSOCIATION_DEF_DATA def_data;  /* association definition        */
} TN3270_ASSOCIATION;


typedef struct tn3270_association_def_data
{
   unsigned char  description[32];        /* resource description          */
   unsigned char  reserve0[16];           /* reserved                      */
   unsigned char  printer_lu_name[8];     /* name of printer LU/pool       */
```

```
   unsigned char  reserv2[8];              /* reserved                        */
} TN3270_ASSOCIATION_DEF_DATA;
```

Data is returned in the form of `tn3270_association` structures.

Supplied Parameters

The application supplies the following parameters:

opcode

`AP_QUERY_TN3270_ASSOCIATION`

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of associations for which data should be returned. To request data for a specific association rather than a range, specify the value `1`. To return as many entries as possible, specify 0; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of associations from which SNAplus2 begins to return data. Specify one of the following values:

`AP_FIRST_IN_LIST`

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *display_lu_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *display_lu_name* parameter.

display_lu_name

Name of the display LU for which association information is required or the name to be used as an index into the list of associations. The display LU name is an EBCDIC string padded on the right with EBCDIC spaces. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. This may be higher than the value supplied for the *buf_size* parameter.

num_entries

The number of entries actually returned.

total_num_entries

Total number of entries that could have been returned. This may be higher than the value supplied for the *num_entries* parameter.

Each entry in the data buffer consists of the following:

tn3270_association.overlay_size

The size of the returned tn3270_association structure (and therefore the offset to the start of the next entry in the data buffer).

tn3270_association.display_lu_name

Name of the display LU associated with the printer LU specified by the *association.printer_lu_name* parameter. This is an EBCDIC string padded on the right with EBCDIC spaces.

tn3270_association_def_data.description

A null-terminated text string that describe the association, as specified in the definition of the association.

tn3270_association_def_data.printer_lu_name

Name of the printer LU associated with the display LU specified by the *association.display_lu_name* parameter. This is an EBCDIC string padded on the right with EBCDIC spaces.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

AP_INVALID_LU_NAME

Indicates one of the following:

- The *list_options* parameter was set to AP_LIST_FROM_NEXT, but the display LU name

was not a valid EBCDIC string.

- The *list_options* parameter was set to AP_LIST_INCLUSIVE, but the display LU name either was not a valid EBCDIC string or did not correspond to an existing association record.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TN3270_DEFAULTS

QUERY_TN3270_DEFAULTS returns information about TN3270 parameters used on all client sessions.

VCB Structure

```
typedef struct query_tn3270_defaults
{
   AP_UINT16        opcode;               /* verb operation code           */
   unsigned char    reserv2;              /* reserved                      */
   unsigned char    format;               /* reserved                      */
   AP_UINT16        primary_rc;           /* primary return code           */
   AP_UINT32        secondary_rc;         /* secondary return code         */
   TN3270_DEFAULTS_DEF_DATA def_data;     /* TN3270 defaults               */
} QUERY_TN3270_DEFAULTS;


typedef struct tn3270_defaults_def_data
{
   AP_UINT16        force_responses;      /* force printer responses?      */
   AP_UINT16        keepalive_method;     /* method for sending keep-alives */
   AP_UINT32        keepalive_interval;   /* interval between keep-alives  */
   unsigned char    reserv2[32];          /* reserved                      */
} TN3270_DEFAULTS_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_TN3270_DEFAULTS

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

def_data.force_responses

Controls client responses on printer sessions. Possible values are:

AP_YES

Requests definite responses.

AP_NO

Request responses matching SNA traffic.

def_data.keepalive_method

Method for sending keep-alive messages. Possible values are:

AP_NONE

Do not send keep-alive messages. This value is the default value.

AP_TN3270_NOP

Send Telnet NOP messages.

AP_TN3270_TM

Send Telnet DO TIMING-MARK messages.

def_data.keepalive_interval

Interval between consecutive keep-alive messages, in seconds.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TP

QUERY_TP returns information about TPs that are currently using a local LU. This verb can be used to obtain information about a specific TP or about multiple TPs, depending on the options used. This verb returns information about current usage of the TPs, not about their definition; use QUERY_TP_DEFINITION to obtain the definition of the TPs.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_tp
{
  AP_UINT16        opcode;                  /* Verb operation code        */
  unsigned char    reserv2;                 /* reserved                   */
  unsigned char    format;                  /* reserved                   */
  AP_UINT16        primary_rc;              /* Primary return code        */
  AP_UINT32        secondary_rc;            /* Secondary return code      */
  unsigned char    *buf_ptr;                /* pointer to buffer          */
  AP_UINT32        buf_size;                /* buffer size                */
  AP_UINT32        total_buf_size;          /* total buffer size required */
  AP_UINT16        num_entries;             /* number of entries          */
  AP_UINT16        total_num_entries;       /* total number of entries    */
  unsigned char    list_options;            /* listing options            */
  unsigned char    reserv3;                 /* reserved                   */
  unsigned char    lu_name[8];              /* LU name                    */
  unsigned char    lu_alias[8];             /* LU alias                   */
  unsigned char    tp_name[64];             /* TP name                    */
} QUERY_TP;

typedef struct tp_data
{
  AP_UINT16        overlay_size;            /* size of returned entry     */
  unsigned char    tp_name[64];             /* TP name                    */
  unsigned char    description[32];          /* resource description       */
  unsigned char    reserv1[16];             /* reserved                   */
  AP_UINT16        instance_limit;          /* maximum instance count     */
  AP_UINT16        instance_count;          /* current instance count     */
  AP_UINT16        locally_started_count;   /* locally started instance   */
                                            /*  count                     */
  AP_UINT16        remotely_started_count;  /* remotely started instance  */
                                            /*  count                     */
```

```
  unsigned char    reserva[20];               /* reserved                   */
} TP_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_TP

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of TPs for which data should be returned. To request data for a specific TP rather than a range, specify the value `1`. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of TPs from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of LU name and TP name.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of LU name and TP name.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

lu_name

LU name. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters. To specify that the LU is identified by its alias rather than its LU name, set this parameter to 8 binary zeros and specify the LU alias in the following parameter. To specify the LU associated with the local CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

lu_alias

Locally defined LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. This parameter is used only if *lu_name* is set to 8 binary zeros; it is ignored otherwise. To specify the LU associated with the local CP (the default LU), set both *lu_name* and *lu_alias* to binary zeros.

tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

tp_data.overlay_size

The size of the returned tp_data structure, and therefore the offset to the start of the next entry in the data buffer.

tp_data.tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters.

tp_data.description

A null-terminated text string describing the TP, as specified in the definition of the TP.

tp_data.instance_limit

Maximum number of concurrently active instances of the specified TP.

tp_data.instance_count

Number of instances of the specified TP that are currently active.

tp_data.locally_started_count

Number of instances of the TP that have been started

locally (by the TP issuing a TP_STARTED verb).

tp_data.remotely_started_count

Number of instances of the TP that have been started remotely (by a received Attach request).

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

AP_INVALID_LU_ALIAS

The supplied *lu_alias* parameter was not valid.

AP_INVALID_LU_NAME

The supplied *lu_name* parameter was not valid.

AP_INVALID_TP_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied name, but the *tp_name* parameter was not valid.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TP_DEFINITION

QUERY_TP_DEFINITION returns information about TPs defined on the SNAplus2 system. This verb can be used to obtain information about a specific TP or about multiple TPs, depending on the options used. It returns information about the definition of the TPs, not about their current usage; use QUERY_TP to obtain the usage information.

VCB Structure

```
typedef struct query_tp_definition
{
   AP_UINT16        opcode;               /* Verb operation code          */
   unsigned char    reserv2;              /* reserved                     */
   unsigned char    format;               /* reserved                     */
   AP_UINT16        primary_rc;           /* Primary return code          */
   AP_UINT32        secondary_rc;         /* Secondary return code        */
   unsigned char    *buf_ptr;             /* pointer to buffer            */
   AP_UINT32        buf_size;             /* buffer size                  */
   AP_UINT32        total_buf_size;       /* total buffer size required   */
   AP_UINT16        num_entries;          /* number of entries            */
   AP_UINT16        total_num_entries;    /* total number of entries      */
   unsigned char    list_options;         /* listing options              */
   unsigned char    reserv3;              /* reserved                     */
   unsigned char    tp_name[64];          /* TP name                      */
} QUERY_TP_DEFINITION;

typedef struct tp_def_summary
{
   AP_UINT16        overlay_size;         /* size of returned entry       */
   unsigned char    tp_name[64];          /* TP name                      */
   unsigned char    description[32];      /* resource description         */
   unsigned char    reserv1[16];          /* reserved                     */
} TP_DEF_SUMMARY;

typedef struct tp_def_detail
{
   AP_UINT16        overlay_size;         /* size of returned entry       */
   unsigned char    tp_name[64];          /* TP name                      */
   TP_CHARS         tp_chars;             /* TP characteristics           */
} TP_DEF_DETAIL;
```

```
typedef struct tp_chars
{
   unsigned char    description[32];        /* resource description          */
   unsigned char    security_list_name[14]  /* security access list name     */
   unsigned char    reserv1[2];             /* reserved                      */
   unsigned char    conv_type;              /* conversation type             */
   unsigned char    security_rqd;           /* security support              */
   unsigned char    sync_level;             /* synchronization level support */
   unsigned char    dynamic_load;           /* dynamic load                  */
   unsigned char    enabled;                /* is the TP enabled?            */
   unsigned char    pip_allowed;            /* program initialization        */
                                            /*  parameters supported         */
   unsigned char    duplex_support;         /* duplex type(s) supported      */
   unsigned char    reserv3[10];            /* reserved                      */
   AP_UINT16        tp_instance_limit;      /* limit on currently active TP  */
                                            /*  instances                    */
   AP_UINT16        tp_data_len;            /* reserved                      */
   unsigned char    tp_data[120];           /* reserved                      */
} TP_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_TP_DEFINITION

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the `sizeof()` function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of TPs for which data should be returned. To request data for a specific TP rather than

a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data, and the level of information required for each entry. Specify the level of information with one of the following values:

AP_SUMMARY

Summary information only.

AP_DETAIL

Detailed information.

Combine this value using a logical OR operation with one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *tp_name* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *tp_name* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters. This parameter is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

tp_def_summary.overlay_size

The size of the returned tp_def_summary structure, and therefore the offset to the start of the next entry in the data buffer.

tp_def_summary.tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters.

tp_def_summary.description

A null-terminated text string describing the TP, as specified in the definition of the TP.

tp_def_detail.tp_chars.security_list_name

Name of the security access list used by this TP (defined using the DEFINE_SECURITY_ACCESS_LIST verb). This

parameter restricts the TP so that only the users named in the specified list can allocate conversations with it.

If this parameter is set to 14 binary zeros, the TP is available for use by any user.

tp_def_detail.overlay_size

The size of the returned `tp_def_detail` structure, and therefore the offset to the start of the next entry in the data buffer.

tp_def_detail.tp_name

TP name. This is a 64-byte string, padded on the right with spaces if the name is shorter than 64 characters.

tp_def_detail.description

A null-terminated text string describing the TP, as specified in the definition of the TP.

tp_def_detail.tp_chars.conv_type

Specifies the type or types of conversation supported by the TP. Possible values are:

`AP_BASIC`

The TP supports only basic conversations.

`AP_MAPPED`

The TP supports only mapped conversations.

`AP_EITHER`

The TP supports either basic or mapped conversations.

tp_def_detail.tp_chars.security_rqd

Specifies the level of conversation security information required to start the TP. Possible values are:

`AP_YES`

A user ID and password are required to start the TP.

`AP_NO`

No security information is required.

tp_def_detail.tp_chars.sync_level

Specifies the values of synchronization level supported by the TP. Possible values are:

AP_NONE

The TP supports only *sync_level* NONE.

AP_CONFIRM_SYNC_LEVEL

The TP supports only *sync_level* CONFIRM.

AP_EITHER

The TP supports either *sync_level* NONE or CONFIRM.

AP_SYNCPT_REQUIRED

The TP supports only *sync_level* SYNCPT (syncpoint is required).

AP_SYNCPT_NEGOTIABLE

The TP supports any of the three *sync_level* values NONE, CONFIRM, and SYNCPT.

tp_def_detail.tp_chars.dynamic_load

Specifies whether the TP can be dynamically loaded. This is set to AP_YES.

tp_def_detail.tp_chars.enabled

Specifies whether the TP can be attached successfully. Possible values are:

AP_YES

TP can be attached.

AP_NO

TP cannot be attached.

tp_def_detail.tp_chars.pip_allowed

Specifies whether the TP can receive Program Initialization Parameters (PIP). Possible values are:

AP_YES

TP can receive PIP.

AP_NO

TP cannot receive PIP.

tp_def_detail.tp_chars.duplex_support

Specifies which conversation duplex types are supported by the TP. Possible values are:

AP_HALF_DUPLEX

The TP supports half-duplex conversations only.

AP_FULL_DUPLEX

The TP supports full-duplex conversations.

AP_EITHER_DUPLEX

The TP supports both half-duplex and full-duplex conversations.

tp_def_detail.tp_chars.tp_instance_limit

Limit on the number of concurrently active TP instances.

tp_def_detail.tp_chars.tp_data_len

Length of the implementation dependent TP data.

tp_def_detail.tp_chars.tp_data

SNAplus2 does not use this parameter (it is set to all zeros).

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_TP_NAME

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from

the supplied name, but the *tp_name* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TP_LOAD_INFO

QUERY_TP_LOAD_INFO returns information about TP load information entries. The buffer contains a number of the variably sized `tp_load_info` structures.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_tp_load_info
{
  AP_UINT16              opcode;               /* Verb operation code        */
  unsigned char          reserv2;              /* reserved                   */
  unsigned char          format;               /* reserved                   */
  AP_UINT16              primary_rc;           /* Primary return code        */
  AP_UINT32              secondary_rc;         /* Secondary return code      */
  unsigned char          *buf_ptr;             /* pointer to buffer          */
  AP_UINT32              buf_size;             /* buffer size                */
  AP_UINT32              total_buf_size;       /* total buffer size required */
  AP_UINT16              num_entries;          /* number of entries          */
  AP_UINT16              total_num_entries;    /* total number of entries    */
  unsigned char          list_options;         /* listing options            */
  unsigned char          reserv3;              /* reserved                   */
  unsigned char          tp_name[64];          /* TP name                    */
  unsigned char          lu_alias[8];          /* LU alias                   */
} QUERY_TP_LOAD_INFO;

typedef struct tp_load_info
{
  AP_UINT16              overlay_size;         /* size of returned entry     */
  unsigned char          tp_name[64];          /* TP name                    */
  unsigned char          lu_alias[8];          /* LU alias                   */
  TP_LOAD_INFO_DEF_DATA  def_data;             /* defined data               */
} TP_LOAD_INFO;

typedef struct tp_load_info_def_data
{
  unsigned char          description[32];      /* Description                */
  unsigned char          reserv1[16];          /* reserved                   */
  unsigned char          user_id[64];          /* User ID                    */
  unsigned char          group_id[64];         /* Group ID                   */
```

```
  unsigned shor          timeout;               /* Timeout value            */
  unsigned char          type;                  /* TP type                  */
  unsigned char          reserv2;               /* reserved                 */
  AP_UINT16              reserv3;               /* reserved                 */
  AP_UINT16              ltv_length;            /* Length of LTV data       */
} TP_LOAD_INFO_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_TP_LOAD_INFO

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof() function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of extra data control blocks for which data should be returned. To request data for a specific resource rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list of TPs from which SNAplus2 should begin to return data. Possible values are:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the combination of TP name and LU alias.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the combination of TP name and LU alias.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

tp_name

TP name to query. This is a 64-byte EBCDIC string, padded on the right with spaces if the name is shorter than 64 characters. Specify all binary zeroes to match on all TPs. This value is ignored if *list_options* is set to AP_FIRST_IN_LIST.

lu_alias

The LU alias to query. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes. Specify all binary zeroes to match on all LUs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

tp_load_info.overlay_size

The size of this overlay, including the LTV data. This size includes padding to ensure that the next overlay falls on a properly aligned memory location.

tp_load_info.tp_name

TP name of the TP load info entry. This is a 64-byte EBCDIC string, padded on the right with spaces if the name is shorter than 64 characters.

tp_load_info.lu_alias

The LU alias of the TP load info entry. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

def_data.description

Description of the TP load info.

def_data.user_id

User ID required to access and run the TP.

def_data.group_id

Group ID required to access and run the TP.

def_data.timeout

Timeout in seconds after the TP is loaded.

def_data.type

Indicates the TP type. Possible values are:

AP_TP_QUEUED

AP_TP_QUEUED_BROADCAST

AP_TP_NON_QUEUED

def_data.ltv_length

Length of the LTV data buffer appended to this structure. Each LTV structure is specified in "TP_LOAD_INFO_LTV" under "DEFINE_TP_LOAD_INFO".

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_TP_NAME

The *tp_name* parameter did not match the name of a defined TP.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined LU alias.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, "Common Return Codes," lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TRACE_FILE

This verb returns information about the files that SNAplus2 uses to record trace data.

This verb may be issued to a running node, or (for client-server trace files only) to a HP-UX client computer on which the SNAplus2 software is running. To obtain a target handle for the client in order to issue this verb, use the CONNECT_NODE verb without specifying a node name; the NOF application must be running on the client.

On Windows clients, client-server tracing is controlled by options in the `sna.ini` file; for more information, see the *HP-UX SNAplus2 Administration Guide*.

VCB Structure

```
typedef struct query_trace_file
{
   AP_UINT16        opcode;                 /* verb operation code       */
   unsigned char    reserv2;                /* reserved                  */
   unsigned char    format;                 /* reserved                  */
   AP_UINT16        primary_rc;             /* primary return code       */
   AP_UINT32        secondary_rc;           /* secondary return code     */
   unsigned char    trace_file_type;        /* type of trace file        */
   unsigned char    dual_files;             /* dual trace files          */
   AP_UINT32        trace_file_size;        /* trace file size           */
   unsigned char    file_name[81];          /* file name                 */
   unsigned char    file_name_2[81];        /* second file name          */
} QUERY_TRACE_FILE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_TRACE_FILE

trace_file_type

The type of trace file. Possible values are:

AP_CS_TRACE

File contains tracing on data transferred across the SNAplus2 LAN between the specified computer and other nodes (activated by the SET_CS_TRACE verb).

AP_BCK_CS_TRACE

File contains tracing on data transferred across the SNAplus2 LAN between a current-level server and back-level computers (activated by the SET_BCK_CS_TRACE verb).

AP_TN_SERVER_TRACE

File contains tracing on the SNAplus2 TN server component.

AP_IPS_TRACE

File contains tracing on kernel components for the specified node (activated by the SET_TRACE_TYPE or ADD_DLC_TRACE verb).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

dual_files

Specifies whether tracing is to one file or to two files. Possible values are:

AP_YES

Tracing is to two files. When the first file reaches the size specified by *trace_file_size*, the second file is cleared, and tracing continues to the second file. When this file then reaches the size specified by *trace_file_size*, the first file is cleared, and tracing continues to the first file. This ensures that tracing can continue for long periods without using excessive disk

space; the maximum space required is approximately twice the value of *trace_file_size*.

AP_NO

Tracing is to one file.

trace_file_size

The maximum size of the trace file. If *dual_files* is set to AP_YES, tracing will switch between the two files when the current file reaches this size. If *dual_files* is set to AP_NO, this parameter is ignored; the file size is not limited.

file_name

Name of the trace file, or of the first trace file if *dual_files* is set to AP_YES. This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero).

If no path is included, the file is stored in the default directory for diagnostics files, /var/opt/sna if a path is included, this is either a full path (starting with a / character) or the path relative to the default directory.

file_name_2

Name of the second trace file; this parameter is used only if *dual_files* is set to AP_YES. This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero).

If no path is included, the file is stored in the default directory for diagnostics files, /var/opt/sna if a path is included, this is either a full path (starting with a / character) or the path relative to the default directory.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_FILE_TYPE

The *trace_file_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_TRACE_TYPE

This verb returns information about the current tracing options for SNAplus2 kernel components. For more information about tracing options, see the *HP-UX SNAplus2 Administration Guide*.

This verb does not return information about DLC line tracing. To do this, use the QUERY_DLC_TRACE verb.

This verb must be issued to a running node.

VCB Structure

```
typedef struct query_trace_type
{
   AP_UINT16        opcode;               /* verb operation code            */
   unsigned char    reserv2;              /* reserved                       */
   unsigned char    format;               /* reserved                       */
   AP_UINT16        primary_rc;           /* primary return code            */
   AP_UINT32        secondary_rc;         /* secondary return code          */
   AP_UINT16        trace_flags;          /* trace flags                    */
   AP_UINT32        truncation_length;    /* truncate each msg to this size */
   AP_UINT16        internal_level;       /* reserved                       */
   AP_UINT32        api_flags;            /* reserved                       */
} QUERY_TRACE_TYPE;
```

Supplied Parameters

The application supplies the following parameter:

opcode

AP_QUERY_TRACE_TYPE

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

secondary_rc

Not used.

trace_flags

The types of tracing currently active. For more information about these trace types, see "SET_TRACE_TYPE".

If no tracing is active, or if tracing of all types is active, this is one of the following values:

AP_NO_TRACE

No tracing.

AP_ALL_TRACE

Tracing of all types.

If tracing is being used on specific interfaces, this parameter is set to one or more values from the list below, combined using a logical OR operation.

AP_APPC_MSG

APPC messages

AP_FM_MSG

FM messages

AP_LUA_MSG

LUA messages

AP_NOF_MSG

NOF messages

AP_MS_MSG

MS messages

AP_PV_MSG

APPC and CPI-C messages for back-level computers

AP_DLPI_MSG

DLPI messages

AP_SDLC_MSG

SDLC messages

`AP_NLI_MSG`

NLI messages

`AP_DLC_MSG`

Node to DLC messages

`AP_NODE_MSG`

Node messages

`AP_SLIM_MSG`

Messages sent between master and backup servers in a client-server system

`AP_DATAGRAM`

Datagram messages

`truncation_length`

The maximum length, in bytes, of the information written to the trace file for each message. If a message is longer than this, SNAplus2 writes only the start of the message to the trace file, and discards the data beyond *`truncation_length`*. This allows you to record the most important information for each message but avoid filling up the file with long messages. A value of zero indicates that trace messages are not truncated.

`api_flags`

This parameter is reserved.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

QUERY_USERID_PASSWORD

QUERY_USERID_PASSWORD returns information about user ID / password pairs for use with APPC and CPI-C conversation security, or about profiles for a defined user ID and password. It can be used to obtain information about a specific user ID / password pair or about multiple pairs, depending on the options used.

This verb returns password information in clear text. Be careful about how you use this verb, to ensure that unauthorized users are not given access to password information.

VCB Structure

```
typedef struct query_userid_password
{
   AP_UINT16         opcode;                /* Verb operation code         */
   unsigned char     reserv2;               /* reserved                    */
   unsigned char     format;                /* reserved                    */
   AP_UINT16         primary_rc;            /* Primary return code         */
   AP_UINT32         secondary_rc;          /* Secondary return code       */
   unsigned char     *buf_ptr;              /* pointer to buffer           */
   AP_UINT32         buf_size;              /* buffer size                 */
   AP_UINT32         total_buf_size;        /* total buffer size required  */
   AP_UINT16         num_entries;           /* number of entries           */
   AP_UINT16         total_num_entries;     /* total number of entries     */
   unsigned char     list_options;          /* listing options             */
   unsigned char     reserv3;               /* reserved                    */
   unsigned char     user_id[10];           /* user ID                     */
} QUERY_USERID_PASSWORD;

typedef struct userid_info
{
   AP_UINT16              overlay_size;     /* size of returned entry      */
   unsigned char          user_id[10];      /* user ID                     */
   USERID_PASSWORD_CHARS  password_chars;   /* password characteristics    */
} USERID_INFO;

typedef struct userid_password_chars
{
   unsigned char          description[32];  /* resource description        */
   unsigned char          reserv2[16];      /* reserved                    */
```

```
   AP_UINT16              profile_count;      /* number of profiles          */
   AP_UINT16              reserv1;            /* reserved                    */
   unsigned char          password[10];       /* password                    */
   unsigned char          profiles[10][10];   /* profiles                    */
} USERID_PASSWORD_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_QUERY_USERID_PASSWORD

overlay_size

For compatability with future releases of SNAplus2, your application must use this field to determine the exact length of the overlay structure returned, and should not rely on the use of the sizeof()function.

buf_ptr

A pointer to a data buffer that SNAplus2 will use to return the requested information.

buf_size

Size of the supplied data buffer.

num_entries

Maximum number of user ID / password pairs for which data should be returned. To request a specific entry rather than a range, specify the value 1. To return as many entries as possible, specify zero; in this case, SNAplus2 will return the maximum number of entries that can be accommodated in the supplied data buffer.

list_options

The position in the list from which SNAplus2 should begin to return data. Specify one of the following values:

AP_FIRST_IN_LIST

Start at the first entry in the list.

AP_LIST_INCLUSIVE

Start at the entry specified by the *user_id* parameter.

AP_LIST_FROM_NEXT

Start at the entry immediately following the entry specified by the *user_id* parameter.

For more information about how the list is ordered and how the application can obtain specific entries from it, see “List Options For QUERY_* Verbs”.

user_id

User ID. This is a 10-byte type-AE EBCDIC string, padded on the right with spaces if the name is shorter than 10 characters. The user ID is ignored if *list_options* is set to AP_FIRST_IN_LIST.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc

AP_OK

buf_size

Length of the information returned in the supplied buffer.

total_buf_size

Returned value indicating the size of buffer that would have been required to return all the list information requested. A value greater than *buf_size* indicates that not all the available entries were returned.

num_entries

Number of entries returned in the data buffer.

total_num_entries

Total number of entries available. A value greater than *num_entries* indicates that not all the available entries were returned.

Each entry in the data buffer consists of the following parameters:

userid_info.overlay_size

The size of the returned userid_info structure, and therefore the offset to the start of the next entry in the data buffer.

userid_info.user_id

User identifier. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

userid_info.password_chars.description

A null-terminated text string describing the user ID and password, as specified in the definition of the user ID and password.

userid_info.password_chars.profile_count

Number of profiles defined for this user.

userid_info.password_chars.password

An encrypted version of the user's password supplied on a DEFINE_USERID_PASSWORD verb. This is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

userid_info.password_chars.profiles

Profiles associated with user. Each of these is a 10-byte type-AE EBCDIC character string, padded on the right with EBCDIC spaces.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_INVALID_USERID

The *list_options* parameter was set to AP_LIST_INCLUSIVE to list all entries starting from the supplied user ID, but the *user_id* parameter was not valid.

AP_INVALID_LIST_OPTION

The *list_options* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

5 NOF API Verbs (REGISTER_INDICATION_SINK to UNREGISTER_INDICATION_SINK)

REGISTER_INDICATION_SINK

REGISTER_INDICATION_SINK registers the NOF application to receive indications of a particular type; for details of the SNAplus2 NOF indications, see Chapter 6, "NOF Indications." The application specifies the required type of indication by its *opcode* parameter; an application can register more than once to accept multiple indication types. Each time an event occurs for which the application has requested indications (for example a change in the configuration of the application's target node or a change in the status of a DLC), SNAplus2 sends the appropriate indication message to the application.

A NOF_STATUS_INDICATION, which indicates status changes for the target node or file, may be returned to an application that has registered for any type of indication. For more information, see "NOF_STATUS_INDICATION".

This verb must always be issued using the asynchronous NOF API entry point, including a callback routine (for more information about the NOF API entry points, see "Asynchronous Entry Point: nof_async"). SNAplus2 uses this callback routine to return the requested indications to the application.

This verb may be issued to different targets depending on the type of indications required, as follows:

- To register for SNA network file indications, the target must be the `sna.net` file.
- To register for server indications, no target is required; the application must specify a null target handle.
- To register for configuration indications relating to domain resources, the target must be the domain configuration file.
- To register for configuration indications relating to node resources, or to register for any other indications, the target may be either a running node or an inactive node on a computer where the SNAplus2 software is running.

VCB Structure

```
typedef struct register_indication_sink
{
```

```
   AP_UINT16        opcode;              /* verb operation code             */
   unsigned char    reserv2;             /* reserved                        */
   unsigned char    format;              /* reserved                        */
   AP_UINT16        primary_rc;          /* primary return code             */
   AP_UINT32        secondary_rc;        /* secondary return code           */
   AP_UINT32        proc_id;             /* reserved                        */
   AP_UINT16        queue_id;            /* reserved                        */
   AP_UINT16        indication_opcode;   /* opcode of indication to be sunk */
} REGISTER_INDICATION_SINK;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_REGISTER_INDICATION_SINK

indication_opcode

The *opcode* parameter of the indication to be returned. SNAplus2 will send this indication to the application's callback routine every time the indication is generated.

To receive configuration indications, specify the value AP_CONFIG_INDICATION. If the target handle specified on the REGISTER_INDICATION_SINK verb identifies the domain configuration file, this value requests an indication each time the file is updated; if the target handle identifies a node, this value requests an indication each time the node's configuration is updated.

To receive SNA network file indications, issue the verb using a target handle that identifies the sna.net file, and specify the value AP_SNA_NET_INDICATION. This value requests an indication each time the file is updated.

For all other indications, specify the *opcode* value for the required indication. For more information, see the descriptions of individual indications in Chapter 6, “NOF Indications.”

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_OP_CODE

One of the following has occurred:

- The *indication_opcode* parameter did not match the *opcode* of any of the SNAplus2 NOF API indications.
- The *indication_opcode* parameter specified an indication type that does not apply to the specified target. If the target handle identifies the domain configuration file, only configuration indications are valid; if the target handle identifies the sna.net file, only SNA network file indications are valid; and if the target handle specifies a running node, all indications except SNA network file indications are valid.

AP_DYNAMIC_LOAD_ALREADY_REGD

The *indication_opcode* parameter was set to a reserved value.

AP_SYNC_NOT_ALLOWED

The application issued REGISTER_INDICATION_SINK using the synchronous NOF entry point. This verb must use the asynchronous entry point.

Appendix A, “Common Return Codes,” lists further secondary return

codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node does not support the function associated with the specified indication, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support the specified indication. For details of the support required for each indication, see the description of each indication in Chapter 6, “NOF Indications.”

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

REMOVE_DLC_TRACE

This verb removes DLC line tracing that was previously specified using ADD_DLC_TRACE. It can be used to remove all tracing on a resource that is currently being traced, to remove the tracing of certain messages from a resource currently being traced, or to remove all DLC line tracing.

VCB Structure

```
typedef struct remove_dlc_trace
{
   AP_UINT16         opcode;                 /* verb operation code        */
   unsigned char     reserv2;                /* reserved                   */
   unsigned char     format;                 /* reserved                   */
   AP_UINT16         primary_rc;             /* primary return code        */
   AP_UINT32         secondary_rc;           /* secondary return code      */
   DLC_TRACE_FILTER  filter;                 /* resource to stop tracing   */
} REMOVE_DLC_TRACE;

typedef struct dlc_trace_filter
{
   unsigned char     resource_type;          /* type of resource           */
   unsigned char     resource_name[8];       /* name of resource           */
   SNA_LFSID         lfsid;                  /* session identifier         */
   unsigned char     message_type            /* type of messages           */
} DLC_TRACE_FILTER;

typedef struct sna_lfsid
{
      union
      {
          AP_UINT16            session_id;
          struct
          {
              unsigned char      sidh;
              unsigned char      sidl;
          } s;
      } uu;
      AP_UINT16       odai;
} SNA_LFSID;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_REMOVE_DLC_TRACE

resource_type The resource type of the trace entry to remove or modify. Possible values are:

AP_ALL_DLC_TRACES

Remove all DLC tracing options, so that no resources are traced. If this option is specified, the remaining parameters on this verb (*resource_name* through *message_type*) are reserved.

AP_ALL_RESOURCES

Remove or modify the tracing options used for tracing all DLCs, ports, and LSs; resources for which DLC_TRACE entries are explicitly defined will continue to be traced.

AP_DLC

Remove or modify tracing for the DLC named in *resource_name*, and for all ports and LSs that use this DLC.

AP_PORT

Remove or modify tracing for the port named in *resource_name*, and for all LSs that use this port.

AP_LS

Remove or modify tracing for the LS named in *resource_name*.

AP_PORT_DEFINED_LS

Modify tracing for the port named in *resource_name* and its defined LSs.

AP_PORT_IMPLICIT_LS

Modify tracing for the port named in *resource_name* and its implicit LSs.

resource_name The name of the DLC, port, or LS, for which tracing is being removed or modified. This parameter is reserved

if *resource_type* is set to AP_ALL_DLC_TRACES or AP_ALL_RESOURCES.

lfsid The Local Form Session Identifier for a session on the specified LS. This is only valid for *resource_type* AP_LS, and indicates that only messages on this session are to be removed. The structure contains the following three values, which are returned in the SESSION_STATS section of a QUERY_SESSION verb:

lfsid.uu.s.sidh Session ID high byte.

lfsid.uu.s.sidl Session ID low byte.

lfsid.odai Origin Destination Assignor Indicator.

message_type The type of messages to trace for the specified resource or session. Set this parameter to AP_TRACE_ALL to remove all messages, or specify one or more of the following values (combined using a logical OR):

AP_TRACE_XID

XID messages

AP_TRACE_SC

Session Control RUs

AP_TRACE_DFC

Data Flow Control RUs

AP_TRACE_FMD

FMD messages

AP_TRACE_SEGS

Non-BBIU segments that do not contain an RH

AP_TRACE_CTL

Messages other then MUs and XIDs

AP_TRACE_NLP

(this message type is currently not used)

AP_TRACE_NC

(this message type is currently not used)

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns one of the following.

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_RESOURCE_TYPE

The *resource_type* parameter specified a value that was not valid.

AP_INVALID_MESSAGE_TYPE

The *message_type* parameter specified a value that was not valid.

AP_INVALID_DLC_NAME

The DLC named in *resource_name* does not have any tracing options set.

AP_INVALID_PORT_NAME

The Port named in *resource_name* does not have any tracing options set.

AP_INVALID_LS_NAME

The LS named in *resource_name* does not have any tracing options set.

AP_INVALID_LFSID_SPECIFIED

The LS named in *resource_name* does not have any tracing options set for the specified LFSID.

AP_INVALID_FILTER_TYPE

The *message_type* parameter specified a message type

that is not currently being traced for the specified resource.

AP_ALL_RESOURCES_NOT_DEFINED

The *resource_type* parameter was set to AP_ALL_RESOURCES, but there is no DLC_TRACE entry defined for tracing options on all resources.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

RESET_SESSION_LIMIT

The RESET_SESSION_LIMIT verb requests SNAplus2 to reset the session limits for a particular LU-LU-mode combination. Sessions may be deactivated as a result of processing this verb.

VCB Structure

```
typedef struct reset_session_limit
{
   AP_UINT16       opcode;                /* verb operation code            */
   unsigned char   reserv2;               /* reserved                       */
   unsigned char   format;                /* reserved                       */
   AP_UINT16       primary_rc;            /* primary return code            */
   AP_UINT32       secondary_rc;          /* secondary return code          */
   unsigned char   lu_name[8];            /* local LU name                  */
   unsigned char   lu_alias[8];           /* local LU alias                 */
   unsigned char   plu_alias[8];          /* partner LU alias               */
   unsigned char   fqplu_name[17];        /* fully qualified partner LU name*/
   unsigned char   reserv3;               /* reserved                       */
   unsigned char   mode_name[8];          /* mode name                      */
   unsigned char   mode_name_select;      /* select mode name               */
   unsigned char   set_negotiable;        /* set max negotiable limit to    */
                                          /*  zero?                         */
   unsigned char   reserv4[8];            /* reserved                       */
   unsigned char   responsible;           /* who is responsible for         */
                                          /*  deactivation                  */
   unsigned char   drain_source;          /* drain source                   */
   unsigned char   drain_target;          /* drain target                   */
   unsigned char   force;                 /* force                          */
   AP_UINT32       sense_data;            /* sense data                     */
} RESET_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_RESET_SESSION_LIMIT

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the LU is defined by its LU alias instead of its LU name, set this parameter to 8 binary zeros.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. It is used only if *lu_name* is set to zeros.

To indicate the LU associated with the CP (the default LU), set both *lu_name* and *lu_alias* to 8 binary zeros.

plu_alias

LU alias of the partner LU.

This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes. To indicate that the partner LU is defined by its fully qualified LU name instead of its LU alias, set this parameter to 8 binary zeros.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. This parameter is used only if the *plu_alias* field is set to zeros; it is ignored if *plu_alias* is specified.

The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Name of the mode for which to reset session limits. This parameter is ignored if *mode_name_select* is set to AP_ALL.

This is an 8-byte alphanumeric type-A EBCDIC string

(starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 bytes.

mode_name_select

Selects whether session limits should be reset on a single specified mode, or on all modes between the local and partner LUs. Possible values are:

AP_ONE

Reset session limits on the mode specified by *mode_name*.

AP_ALL

Reset session limits on all modes.

set_negotiable

Specifies whether the maximum negotiable session limit for this LU-LU-mode combination should be reset to zero. (The current limit may be the limit specified for the mode, or may have been changed by *initialize_session_limit* or *change_session_limit*). Possible values are:

AP_YES

Reset the maximum negotiable session limit for this LU-LU-mode combination to zero so that sessions cannot be activated until it is changed by INITIALIZE_SESSION_LIMIT.

AP_NO

Leave the maximum negotiable session limit unchanged.

responsible

Indicates whether the source (local) or target (partner) LU is responsible for deactivating sessions after the session limit is reset. Possible values are:

AP_SOURCE

The local LU is responsible for deactivating sessions.

AP_TARGET

The partner LU is responsible for deactivating sessions.

drain_source

Specifies whether the source LU satisfies waiting session requests before deactivating a session. Possible values are:

AP_YES

Waiting session requests are satisfied.

AP_NO

Waiting session requests are not satisfied.

drain_target

Specifies whether the target LU satisfies waiting session requests before deactivating a session. Possible values are:

AP_YES

Waiting session requests are satisfied.

AP_NO

Waiting session requests are not satisfied.

force

Specifies whether session limits will be set to zero even if CNOS negotiation fails. Possible values are:

AP_YES

Session limits will be set to zero.

AP_NO

Session limits will not be set to zero if CNOS negotiation fails.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Possible values are:

AP_FORCED

The session limits were set to zero even though CNOS negotiation failed.

AP_AS_NEGOTIATED

The session limits were changed, but one or more values were negotiated by the partner LU.

AP_AS_SPECIFIED

The session limits were changed as requested, without being negotiated by the partner LU.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_EXCEEDS_MAX_ALLOWED

A SNAplus2 internal error occurred.

AP_INVALID_LU_ALIAS

The *lu_alias* parameter did not match any defined local LU alias.

AP_INVALID_LU_NAME

The *lu_name* parameter did not match any defined local LU name.

AP_INVALID_MODE_NAME

The *mode_name* parameter did not match any defined mode name.

AP_INVALID_PLU_NAME

The *fqplu_name* parameter did not match any defined partner LU name.

AP_INVALID_MODE_NAME_SELECT

The *mode_name_select* parameter was not set to a valid value.

AP_INVALID_DRAIN_SOURCE

The *drain_source* parameter was not set to a valid value.

AP_INVALID_DRAIN_TARGET

The *drain_target* parameter was not set to a valid value.

AP_INVALID_FORCE

The *force* parameter was not set to a valid value.

AP_INVALID_RESPONSIBLE

The *responsible* parameter was not set to a valid value.

AP_INVALID_SET_NEGOTIABLE

The *set_negotiable* parameter was not set to a valid value.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_MODE_RESET

No sessions are currently active for this LU-LU-mode combination. Use INITIALIZE_SESSION_LIMIT instead of RESET_SESSION_LIMIT to specify the limits.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Session Allocation Error

If the verb does not execute because of a session allocation error, SNAplus2 returns the following parameters:

primary_rc	AP_ALLOCATION_ERROR
secondary_rc	AP_ALLOCATION_FAILURE_NO_RETRY A session could not be allocated because of a condition that requires corrective action. Check the *sense_data* parameter and any logged messages to determine the reason for the failure, and take any action required. Do not attempt to retry the verb until the condition has been corrected.
sense_data	The SNA sense data associated with the allocation failure.

Returned Parameters: CNOS Processing Errors

If the verb does not execute because of an error, SNAplus2 returns the following parameters.

primary_rc	AP_CONV_FAILURE_NO_RETRY The session limits could not be changed because of a condition that requires action (such as a configuration mismatch or a session protocol error). Check the SNAplus2 log file for information about the error condition, and correct it before retrying this verb.
primary_rc	AP_CNOS_PARTNER_LU_REJECT The verb failed because SNAplus2 failed to negotiate the session limits with the partner. Check configuration at both the local LU and partner LU.
secondary_rc	AP_CNOS_COMMAND_RACE_REJECT The verb failed because the specified mode was being accessed by another administration program (or internally by the SNAplus2 software) for session activation or deactivation, or for session limit processing. The application should retry the verb, preferably after a timeout to allow the race condition to be cleared.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_BCK_CS_TRACE

This verb specifies options for back-level client-server tracing (tracing of data sent between a current-level SNAplus2 server and a back-level server or client). It is used only when you are in the processs of migrating a client-server SNAplus2 system to a new release of the software, so that one or more servers are running the current-level software and providing support for computers running the back-level software. For more information about the migration process and on support for back-level computers, see the *HP-UX SNAplus2 Upgrade Guide*.

This verb must be issued to a running node on a computer where the back-level support software is running (for more information, see the *HP-UX SNAplus2 Upgrade Guide*).

VCB Structure

```
typedef struct set_bck_cs_trace
{
   AP_UINT16        opcode;                 /* verb operation code        */
   unsigned char    reserv2;                /* reserved                   */
   unsigned char    format;                 /* reserved                   */
   AP_UINT16        primary_rc;             /* primary return code        */
   AP_UINT32        secondary_rc;           /* secondary return code      */
   AP_UINT32        trace_flags;            /* trace flags                */
   unsigned char    init_flags;             /* YES if initializing flags  */
   unsigned char    set_flags;              /* YES if setting flags       */
                                            /*  NO if unsetting flags     */
} SET_BCK_CS_TRACE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_SET_BCK_CS_TRACE

trace_flags The types of tracing required. To turn off all tracing, or to turn on tracing of all types, specify one of the following values:

AP_BCK_NO_TRACE

No tracing.

AP_BCK_ALL_TRACE

Tracing of all types.

To activate tracing on specific message types, select one or more values from the list below, combined using a logical OR operation.

AP_BCK_UDP_TRACE

UDP messages

AP_BCK_TCP_TRACE

TCP messages

AP_BCK_UDP_TEXT

Text strings describing UDP events

AP_BCK_TCP_TEXT

Text strings describing TCP events

AP_BCK_TEXT

Text strings describing other events

AP_BCK_MS_TRACE

Tracing on TRANSFER_MS_DATA verbs (passed to the MS library on the server)

init_flags Specifies whether to initialize tracing (define the tracing state at all interfaces), or to change the state of tracing at one or more interfaces (leaving the others unchanged). Possible values are:

AP_YES

Tracing is being initialized. The *trace_flags* parameter defines the required state of tracing at all interfaces.

AP_NO

Tracing is being changed. The *trace_flags* parameter defines the interfaces where tracing is being activated or deactivated; other interfaces will not be affected.

set_flags If *init_flags* is set to AP_NO, this parameter specifies whether tracing is to be activated or deactivated at the

requested interfaces. Possible values are:

AP_YES

Tracing is to be activated at the interfaces specified by the *trace_flags* parameter.

AP_NO

Tracing is to be deactivated at the interfaces specified by the *trace_flags* parameter.

If *init_flags* is set to AP_YES, this parameter is ignored.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_INVALID_TARGET

The back-level support software is not running on the target server.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_BUFFER_AVAILABILITY

This verb specifies the amount of STREAMS buffers that SNAplus2 can use at any one time. This allows the node to make efficient use of the buffers available, and allows you to ensure that buffers are available for other processes on the HP-UX computer.

VCB Structure

```
typedef struct set_buffer_availability
{
   AP_UINT16        opcode;              /* verb operation code            */
   unsigned char    reserv2;             /* reserved                       */
   unsigned char    format;              /* reserved                       */
   AP_UINT16        primary_rc;          /* primary return code            */
   AP_UINT32        secondary_rc;        /* secondary return code          */
   AP_UINT32        buf_avail;           /* maximum buffer space available  */
} SET_BUFFER_AVAILABILITY;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_SET_BUFFER_AVAILABILITY
buf_avail	The maximum amount of STREAMS buffer space available, in bytes.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc	AP_OK
secondary_rc	Not used.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_CENTRAL_LOGGING

This verb specifies whether SNAplus2 log messages are sent to a central file from all servers, or to a separate file on each server. For more information, see "SET_LOG_FILE".

This verb must be issued to the node that is currently acting as the central logger; for information about accessing this node, see "CONNECT_NODE".

VCB Structure

```
typedef struct set_central_logging
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* primary return code          */
   AP_UINT32        secondary_rc;          /* secondary return code        */
   unsigned char    enabled;               /* is central logging enabled?  */
} SET_CENTRAL_LOGGING;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_SET_CENTRAL_LOGGING

enabled Specify whether central logging is enabled or disabled. Possible values are:

AP_YES

Central logging is enabled. All log messages are sent to a single file on the node currently acting as the central logger.

AP_NO

Central logging is disabled. Log messages from each server are sent to a file on that server (specified using the SET_LOG_FILE verb).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_NOT_CENTRAL_LOGGER

The verb was issued to a node that is not the central logger.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_CS_TRACE

This verb specifies tracing options for data sent between computers on the SNAplus2 LAN. For more information about tracing options, see the *HP-UX SNAplus2 Administration Guide*.

This verb may be issued to a running node, or to a HP-UX client computer on which the SNAplus2 software is running. To obtain a target handle for the client in order to issue this verb, use the CONNECT_NODE verb without specifying a node name; the NOF application must be running on the client.

On Windows clients, client-server tracing is controlled by options in the `sna.ini` file; for more information, see the *HP-UX SNAplus2 Administration Guide*.

VCB Structure

```
typedef struct set_cs_trace
{
   unsigned short   opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   unsigned short   primary_rc;       /* primary return code               */
   unsigned long    secondary_rc;     /* secondary return code             */
   unsigned char    dest_sys[64];     /* node to which messages are traced  */
   unsigned char    reserv4[4];       /* reserved                          */
   unsigned short   trace_flags;      /* trace flags                       */
   unsigned short   trace_direction;  /* direction (send/rcv/both) to trace */
} SET_CS_TRACE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_SET_CS_TRACE

dest_sys The server name for which tracing is required. This is an ASCII string, padded on the right with spaces if the name is shorter than 64 characters.

To manage tracing on messages flowing between the computer to which this verb is issued (identified by the

target_handle parameter on the NOF API call) and one other server on the LAN, specify the name of the other server here. Tracing on messages flowing to and from other computers on the LAN will be unchanged; in particular, you can issue two SET_CS_TRACE verbs to activate tracing between the same target computer and two different destination servers.

To manage tracing on messages flowing between the computer to which this verb is issued (identified by the *target_handle* parameter on the NOF API call) and all other servers and clients on the LAN, set this parameter to 64 ASCII space characters. The options you specify on this verb override any previous settings for tracing to specific computers (identified by *dest_sys* on the previous verbs).

trace_flags The types of tracing required. To turn off all tracing, or to turn on tracing of all types, specify one of the following values:

AP_NO_TRACE

No tracing.

AP_ALL_TRACE

Tracing of all types.

To activate tracing on specific message types, select one or more values from the list below, combined using a logical OR operation.

AP_CS_ADMIN_MSG

Internal messages relating to client-server topology

AP_CS_DATAGRAM

Datagram messages

AP_CS_DATA

Data messages

trace_direction Specifies the direction(s) in which tracing is required. This parameter is ignored if *trace_flags* is set to AP_NO_TRACE. Possible values are:

AP_CS_SEND

Trace messages flowing from the target computer to the computer defined by *dest_sys*.

AP_CS_RECEIVE

Trace messages flowing from the computer defined by *dest_sys* to the target computer.

AP_CS_BOTH

Trace messages flowing in both directions.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_NAME_NOT_FOUND

The server specified by the *dest_sys* parameter did not exist or was not started.

AP_LOCAL_SYSTEM

The server specified by the *dest_sys* parameter is the same as the target node to which this verb was issued.

AP_INVALID_TRC_DIRECTION

The *trace_direction* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_GLOBAL_LOG_TYPE

This verb specifies the types of information that SNAplus2 records in log files. It specifies default values that are used on all servers; SET_LOG_TYPE can then be used to override these defaults on a particular server. For more information about log file, see "SET_LOG_FILE".

SNAplus2 logs messages for the following types of event:

Problem	An abnormal event that degrades the system in a way perceptible to a user (such as abnormal termination of a session).
Exception	An abnormal event that degrades the system but that is not immediately perceptible to a user (such as a resource shortage), or an event that does not degrade the system but may indicate the cause of later exceptions or problems (such as receiving an unexpected message from the remote system).
Audit	A normal event (such as starting a session).

Problem and exception messages are logged to the error log file; audit messages are logged to the audit log file. Problem messages are always logged and cannot be disabled, but you can specify whether to log each of the other two message types. For each of the two files (audit and error), you can specify whether to use succinct logging (including only the text of the message and a summary of the message source) or full logging (including full details of the message source, cause, and any action required).

This verb must be issued to the node that is currently acting as the central logger; for information about accessing this node, see "CONNECT_NODE".

VCB Structure

```
typedef struct set_global_log_type
{
   AP_UINT16       opcode;            /* verb operation code              */
   unsigned char   reserv2[2];        /* reserved                         */
   unsigned char   format;            /* reserved                         */
   AP_UINT16       primary_rc;        /* primary return code              */
```

```
   AP_UINT32       secondary_rc;       /* secondary return code              */
   unsigned char   audit;              /* audit logging on or off            */
   unsigned char   exception;          /* exception logging on or off        */
   unsigned char   succinct_audits;    /* use succinct logging in audit file?*/
   unsigned char   succinct_errors;    /* use succinct logging in error file?*/
} SET_GLOBAL_LOG_TYPE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_SET_GLOBAL_LOG_TYPE

audit

Specify whether audit messages are recorded. Possible values are:

AP_YES

Audit messages are recorded.

AP_NO

Audit messages are not recorded.

AP_LEAVE_UNCHANGED

Leave audit logging unchanged from the existing definition. (The initial default, when the SNAplus2 software is started, is that audit messages are not recorded.)

exception

Specify whether exception messages are recorded. Possible values are:

AP_YES

Exception messages are recorded.

AP_NO

Exception messages are not recorded.

AP_LEAVE_UNCHANGED

Leave exception logging unchanged from the existing

definition. (The initial default, when the SNAplus2 software is started, is that exception messages are recorded.)

succinct_audits

Specifies whether to use succinct logging or full logging in the audit log file. Possible values are:

AP_YES

Succinct logging: each message in the log file contains a summary of the message header information (such as the message number, log type, and system name) and the message text string and parameters. To obtain more details of the cause of the log and any action required, you can use the snaphelp utility.

AP_NO

Full logging: each message in the log file includes a full listing of the message header information, the message text string and parameters, and additional information about the cause of the log and any action required.

AP_LEAVE_UNCHANGED

Use the value (succinct logging or full logging) specified for this parameter on the previous SET_GLOBAL_LOG_TYPE verb. The initial default, before any SET_GLOBAL_LOG_TYPE verb has been issued, is to use succinct logging.

If you are using central logging, the choice of succinct or full logging for messages from all computers is determined by the setting of this parameter on the server acting as the central logger; this setting may either be from the SET_GLOBAL_LOG_TYPE verb, or from a SET_LOG_TYPE verb issued to that server to override the default.

succinct_errors

Specifies whether to use succinct logging or full logging in the error log file; this applies to both exception logs and problem logs. The allowed values and their meanings are the same as for the *succinct_audits* parameter.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_NOT_CENTRAL_LOGGER

The verb was issued to a node that is not the central logger.

AP_INVALID_SUCCINCT_SETTING

The *succinct_audits* or *succinct_errors* parameter was not set to a valid value.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_KERNEL_MEMORY_LIMIT

This verb specifies a limit on the amount of kernel memory that SNAplus2 can use at any one time. This allows you to ensure that memory is available for other processes on the HP-UX computer.

You can also specify the kernel memory limit when starting the SNAplus2 software (for more information, see the *HP-UX SNAplus2 Administration Guide*). This verb overrides the limit, if any, specified when starting the SNAplus2 software.

VCB Structure

```
typedef struct set_kernel_memory_limit
{
   AP_UINT16       opcode;           /* verb operation code                */
   unsigned char   reserv2;          /* reserved                           */
   unsigned char   format;           /* reserved                           */
   AP_UINT16       primary_rc;       /* primary return code                */
   AP_UINT32       secondary_rc;     /* secondary return code              */
   AP_UINT32       limit;            /* kernel memory limit, 0 => no limit  */
} SET_KERNEL_MEMORY_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode	AP_SET_KERNEL_MEMORY_LIMIT
limit	The maximum amount of kernel memory that SNAplus2 should use at any time, in bytes. If a SNAplus2 component attempts to allocate kernel memory that would take the total amount of memory currently allocated to SNAplus2 components above this limit, the allocation attempt will fail. To remove the limit set by a previous SET_KERNEL_MEMORY_LIMIT verb, specify zero.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_LOG_FILE

This verb manages a file that SNAplus2 uses to record log messages. It allows you to do the following:

- Specify the file used to record log messages, and the backup file (to which log information is copied).
- Specify the maximum log file size (when the log file reaches this size, SNAplus2 copies log information to the backup file and resets the log file).
- Copy the current contents of the log file to the backup file, and optionally delete the current file.

You can record audit log and error log messages in separate files, or record both types of messages in the same file.

VCB Structure

```
typedef struct set_log_file
{
   AP_UINT16       opcode;                 /* verb operation code          */
   unsigned char   reserv2;                /* reserved                     */
   unsigned char   format;                 /* reserved                     */
   AP_UINT16       primary_rc;             /* primary return code          */
   AP_UINT32       secondary_rc;           /* secondary return code        */
   unsigned char   log_file_type;          /* type of log file             */
   unsigned char   action;                 /* reset and/or backup existing */
   unsigned char   file_name[81];          /* file name                    */
   unsigned char   backup_file_name[81];   /* backup file                  */
   AP_UINT32       file_size;              /* log file size                */
} SET_LOG_FILE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_SET_LOG_FILE

log_file_type

The type of log file being managed. Possible values are:

AP_AUDIT_FILE

Audit log file (audit messages only).

AP_ERROR_FILE

Error log file (problem and exception messages).

To record both types of messages in the same file, issue two SET_LOG_FILE verbs for the same file name, specifying AP_AUDIT_FILE on one verb and AP_ERROR_FILE on the other.

action

The action to be taken on the log file. Specify one of the following values:

AP_NO_FILE_ACTION

Use the file specified in the *file_name* parameter as the log file, and the file specified in the *backup_file_name* parameter as the backup file. After this verb completes successfully, all log messages of the type defined by *log_file_type* are written to the new log file. The log file that was used before this verb is issued, if any, is left unchanged.

AP_DELETE_FILE

Delete the contents of the current log file.

AP_BACKUP_FILE

Copy the contents of the current log file to the backup file, and then delete the contents of the current file.

file_name

Name of the new log file.

To create the file in the default directory for diagnostics files, /var/opt/sna, specify the file name with no path. To create the file in a different directory, specify either a full path or the path relative to the default directory. If you include the path, ensure that it is a valid path (either relative to the application's working directory or a full path) on any computer to which this verb is

issued.

This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero). To continue logging to the file specified on a previous SET_LOG_FILE verb, specify a null string.

backup_file_name

Name of the backup log file. When the log file reaches the size specified by *file_size* below, SNAplus2 copies the current contents to the backup file and then clears the log file. You can also request a backup at any time using the action parameter above.

To create the file in the default directory for diagnostics files, /var/opt/sna, specify the file name with no path. To create the file in a different directory, specify either a full path or the path relative to the default directory. If you include the path, ensure that it is a valid path (either relative to the application's working directory or a full path) on any computer to which this verb is issued.

This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero). To continue using the backup file specified on a previous SET_LOG_FILE verb, specify a null string.

file_size

The maximum size of the log file specified by *log_file_type*. When a message written to the file causes the file size to exceed this limit, SNAplus2 copies the current contents of the log file to the backup log file and clears the log file. This means that the maximum amount of disk space taken up by log files is approximately twice *file_size*.

To continue using the file size specified on a previous SET_LOG_FILE verb, set this parameter to zero. The initial default value, before any SET_LOG_FILE verb has been issued, is 1,000,000 bytes. A value of zero indicates “continue using the existing file size” and not “no limit”.

You may need to increase the size of the log files

according to the size of the SNAplus2 client-server network, to allow for the volume of log information generated in larger systems. In particular, consider increasing the log file size to allow for the following:

- Large numbers of clients or users (since a single communications link failure may result in large numbers of logs on the server relating to session failures)
- Activating audit logging as well as exception logging
- Using central logging instead of distributed logging
- Using full logging instead of succinct logging.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_FILE_ACTION

The *action* parameter was not set to a valid value.

AP_INVALID_FILE_TYPE

The *log_file_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_LOG_TYPE

This verb specifies the types of information that SNAplus2 records in log files on a particular server. It can be used to override the default settings specified on SET_GLOBAL_LOG_TYPE, or to remove the override so that this server reverts to using the default settings. For more information about log files, see "SET_LOG_FILE".

SNAplus2 logs messages for the following types of event:

Problem	An abnormal event that degrades the system in a way perceptible to a user (such as abnormal termination of a session).
Exception	An abnormal event that degrades the system but that is not immediately perceptible to a user (such as a resource shortage), or an event that does not degrade the system but may indicate the cause of later exceptions or problems (such as receiving an unexpected message from the remote system).
Audit	A normal event (such as starting a session).

Problem and exception messages are logged to the error log file; audit messages are logged to the audit log file. Problem messages are always logged and cannot be disabled, but you can specify whether to log each of the other two message types. For each of the two files (audit and error), you can specify whether to use succinct logging (including only the text of the message and a summary of the message source) or full logging (including full details of the message source, cause, and any action required).

VCB Structure

```
typedef struct set_log_type
{
   AP_UINT16       opcode;            /* verb operation code              */
   unsigned char   reserv2;           /* reserved                         */
   unsigned char   format;            /* reserved                         */
   AP_UINT16       primary_rc;        /* primary return code              */
   AP_UINT32       secondary_rc;      /* secondary return code            */
   unsigned char   override;          /* override global defaults?        */
   unsigned char   audit;             /* audit logging on or off          */
```

```
   unsigned char   exception;          /* exception logging on or off       */
   unsigned char   succinct_audits;    /* use succinct logging in audit file?*/
   unsigned char   succinct_errors;    /* use succinct logging in error file?*/
} SET_LOG_TYPE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_SET_LOG_TYPE

override

Specifies whether this verb is being used to override the global log types specified on SET_GLOBAL_LOG_TYPE, or to revert to using these defaults. Possible values are:

AP_YES

Override the global log types. The log types to be used on this server are specified by the *audit* and *exception* parameters below, and the choice of succinct or full logging is specified by the *succinct_** parameters below.

AP_NO

Revert to using the global log types. The *audit*, *exception*, and *succinct_** parameters below are ignored.

audit

Specify whether audit messages are recorded on this server. Possible values are:

AP_YES

Audit messages are recorded.

AP_NO

Audit messages are not recorded.

AP_LEAVE_UNCHANGED

Leave audit logging unchanged from the existing definition.

exception

Specify whether exception messages are recorded on this server. Possible values are:

AP_YES

Exception messages are recorded.

AP_NO

Exception messages are not recorded.

AP_LEAVE_UNCHANGED

Leave exception logging unchanged from the existing definition.

succinct_audits

Specifies whether to use succinct logging or full logging in the audit log file on this server. Possible values are:

AP_YES

Succinct logging: each message in the log file contains a summary of the message header information (such as the message number, log type, and system name) and the message text string and parameters. To obtain more details of the cause of the log and any action required, you can use the snaphelp utility.

AP_NO

Full logging: each message in the log file includes a full listing of the message header information, the message text string and parameters, and additional information about the cause of the log and any action required.

AP_LEAVE_UNCHANGED

Leave succinct logging or full logging unchanged from the existing definition.

If you are using central logging, the choice of succinct or full logging for messages from all computers is determined by the setting of this parameter on the server acting as the central logger; this setting may

either be from the SET_GLOBAL_LOG_TYPE verb, or from a SET_LOG_TYPE verb issued to that server to override the default.

succinct_errors

Specifies whether to use succinct logging or full logging in the error log file on this server; this applies to both exception logs and problem logs. The allowed values and their meanings are the same as for the *succinct_audits* parameter.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_SUCCINCT_SETTING

The *succinct_audits* or *succinct_errors* parameter was not set to a valid value.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_PROCESSING_MODE

This verb specifies how the NOF application interacts with the target node, configuration file, or SNA network data file: whether the application has read-only access or read/write access, and whether the application has exclusive access to the domain configuration file so that other applications cannot access it.

The target node or file is specified by the *target_handle* parameter on the NOF API call; the application obtains this parameter from the verb CONNECT_NODE (for a node) or OPEN_FILE (for a file). For more information about the use of this parameter, see "Description of the NOF API Entry Points".

This verb may be issued to the domain configuration file, to the sna.net file, or to a running node. The valid processing modes that can be set with this verb depend on the target type.

VCB Structure

```
typedef struct set_processing_mode
{
   AP_UINT16        opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   AP_UINT16        primary_rc;       /* primary return code               */
   AP_UINT32        secondary_rc;     /* secondary return code             */
   AP_UINT32        mode;             /* new mode to be set for this handle */
} SET_PROCESSING_MODE;
```

Supplied Parameters

opcode AP_SET_PROCESSING_MODE

mode Requested mode for this target handle. The mode cannot be changed while any previous verbs issued using this target handle are still outstanding. Possible values are:

AP_MODE_READ_ONLY

Read-only mode: the application will use only

QUERY_* verbs, which do not modify the configuration. This option can be used with either a file or a node as the target.

AP_MODE_READ_WRITE

Read / write mode: the application may use any NOF API verbs. This option can be used with either a file or a node as the target.

AP_MODE_COMMIT

Commit mode: the application has exclusive read/write access to the target file, so that other applications cannot access it until this application releases it. This option can be used only with the domain configuration file as the target.

This mode is intended for issuing a series of connected verbs to a file (such as a series of DEFINE verbs for related components). The application should complete the series of verbs as quickly as possible and then reset its processing mode to one of the other options, in order to release the file so that other NOF API applications or SNAplus2 components can access it.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PROC_MODE

The *mode* parameter was not set to a valid value.

AP_INVALID_TARGET_MODE

The *mode* parameter was not valid for the selected target.

Appendix A, "Common Return Codes," lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_FILE_UNAVAILABLE

The application specified commit mode, but was unable to get exclusive access to the required configuration file. This may be because another application is accessing the file in commit mode.

AP_VERB_IN_PROGRESS

The processing mode for the specified target handle cannot be changed because a previous verb issued for this handle is still outstanding. All verbs for the target handle must be completed before attempting to change the processing mode.

AP_NOT_MASTER

The processing mode cannot be changed to AP_MODE_READ_WRITE or AP_MODE_COMMIT because the target handle specifies a file (either the domain configuration file or the SNA network data file) on a backup server that is no longer acting as the master server. Changes to the running configuration file can be made only to the copy of this file on the master (so that they will be distributed to other servers); other copies of the file can be accessed only in read-only mode. If the application needs to use read/write or commit mode, it should issue CLOSE_FILE for this target handle, and then reissue OPEN_FILE to access the file on the new

master server.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_TN_SERVER_TRACE

This verb specifies tracing options for the SNAplus2 TN server component.

This verb must be issued to a running node.

VCB Structure

```
typedef struct set_tn_server_trace
{
   AP_UINT16        opcode;                    /* verb operation code      */
   unsigned char    reserv2;                   /* reserved                 */
   unsigned char    format;                    /* reserved                 */
   AP_UINT16        primary_rc;                /* primary return code      */
   AP_UINT32        secondary_rc;              /* secondary return code    */
   AP_UINT16        trace_flags;               /* trace flags              */
} SET_TN_SERVER_TRACE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_SET_TN_SERVER_TRACE

trace_flags The types of tracing required. To turn off all tracing, or to turn on tracing of all types, specify one of the following values:

AP_TN_SERVER_NO_TRACE

No tracing.

AP_TN_SERVER_ALL_TRACE

Tracing of all types.

To activate tracing on specific message types, select one or more values from the list below, combined using a logical OR operation.

AP_TN_SERVER_TRC_TCP

TCP/IP interface tracing: messages between TN server and TN3270 clients

AP_TN_SERVER_TRC_FM

Node interface tracing: internal control messages, and messages between TN server and TN3270 clients (in internal format)

AP_TN_SERVER_TRC_CFG

Configuration message tracing: messages relating to the configuration of TN server

AP_TN_SERVER_TRC_NOF

Internal node operator function (NOF) tracing: trace NOF requests made by TN server.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_TRACE_FILE

This verb specifies the name of a file that SNAplus2 uses to record trace data.

If you issue a second SET_TRACE_FILE verb specifying a new file for the same file type, all subsequent trace information will be written to the new file; the existing file is not removed, but further information will not be written to it. If you issue a second SET_TRACE_FILE verb for the same trace file, this resets the file (discarding trace information that was written to the file before the second verb).

This verb may be issued to a running node, or (for client-server trace files only) to a HP-UX client computer on which the SNAplus2 software is running. To obtain a target handle for the client in order to issue this verb, use the CONNECT_NODE verb without specifying a node name; the NOF application must be running on the client.

On Windows clients, client-server tracing is controlled by options in the `sna.ini` file. For more information, see the *HP-UX SNAplus2 Administration Guide*.

VCB Structure

```
typedef struct set_trace_file
{
   AP_UINT16        opcode;                    /* verb operation code    */
   unsigned char    reserv2;                   /* reserved               */
   unsigned char    format;                    /* reserved               */
   AP_UINT16        primary_rc;                /* primary return code    */
   AP_UINT32        secondary_rc;              /* secondary return code  */
   unsigned char    trace_file_type;           /* type of trace file     */
   unsigned char    dual_files;                /* dual trace files       */
   AP_UINT32        trace_file_size;           /* trace file size        */
   unsigned char    file_name[81];             /* file name              */
   unsigned char    file_name_2[81];           /* second file name       */
} SET_TRACE_FILE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_SET_TRACE_FILE

trace_file_type The type of trace file. Possible values are:

AP_CS_TRACE

File contains tracing on data transferred across the SNAplus2 LAN between the specified computer and other nodes (activated by the SET_CS_TRACE verb).

AP_BCK_CS_TRACE

File contains tracing on data transferred across the SNAplus2 LAN between a current-level server and back-level computers. This type of tracing is activated by the SET_BCK_CS_TRACE verb.

AP_TN_SERVER_TRACE

File contains tracing on the SNAplus2 TN server component.

AP_IPS_TRACE

File contains tracing on kernel components for the specified node (activated by the SET_TRACE_TYPE or ADD_DLC_TRACE verb).

dual_files Specifies whether tracing is to one file or to two files. Possible values are:

AP_YES

Tracing is to two files. When the first file reaches the size specified by *trace_file_size*, the second file is cleared, and tracing continues to the second file. When this file then reaches the size specified by *trace_file_size*, the first file is cleared, and tracing continues to the first file. This ensures that tracing can continue for long periods without using excessive disk space; the maximum space required is approximately twice the value of *trace_file_size*.

AP_NO

Tracing is to one file.

AP_LEAVE_UNCHANGED

Leave the *dual_files* setting unchanged from the

existing definition. (The initial default, when the SNAplus2 software is started, is to use two files.)

trace_file_size The maximum size of the trace file, in bytes. To continue using the existing file size definition, specify zero.

If *dual_files* is set to AP_YES, tracing will switch between the two files when the current file reaches this size. If *dual_files* is set to AP_NO, this parameter is ignored; the file size is not limited.

You may need to increase the size of the trace files according to the size of the SNAplus2 client-server network, to allow for the volume of trace information generated in larger systems. In particular, consider increasing the trace file size on a server to allow for large numbers of clients or users accessing the server.

file_name Name of the trace file, or of the first trace file if *dual_files* is set to AP_YES. To continue using the file name specified on a previous SET_TRACE_FILE verb, set this parameter to a null string.

To create the file in the default directory for diagnostics files, /var/opt/sna, specify the file name with no path. To create the file in a different directory, specify either a full path or the path relative to the default directory. If you include the path, ensure that it is a valid path (either relative to the application's working directory or a full path) on any computer to which this verb is issued.

This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero).

file_name_2 Name of the second trace file; this parameter is used only if *dual_files* is set to AP_YES. To continue using the file name specified on a previous *set_trace_file* verb, set this parameter to a null string.

To create the file in the default directory for diagnostics files, /var/opt/sna, specify the file name with no path. To create the file in a different directory, specify either a full path or the path relative to the default directory. If you include the path, ensure that it is a valid path

(either relative to the application's working directory or a full path) on any computer to which this verb is issued.

This parameter is an ASCII string of 1-80 characters, followed by a NULL character (binary zero).

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_FILE_NAME

The *file_name* or *file_name_2* parameter was not set to a valid HP-UX file name, or *file_name_2* was not specified when changing from a single trace file to dual trace files.

AP_INVALID_FILE_TYPE

The *trace_file_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

SET_TRACE_TYPE

This verb specifies tracing options for SNAplus2 kernel components. You can use this verb to specify the state of tracing (on or off) at all interfaces, or to turn tracing on or off at specific interfaces (leaving tracing at other interfaces unchanged). For more information about tracing options, see the *HP-UX SNAplus2 Administration Guide*.

This verb does not control DLC line tracing. To do this, use the ADD_DLC_TRACE verb.

This verb must be issued to a running node.

VCB Structure

```
typedef struct set_trace_type
{
   AP_UINT16        opcode;               /* verb operation code            */
   unsigned char    reserv2;              /* reserved                       */
   unsigned char    format;               /* reserved                       */
   AP_UINT16        primary_rc;           /* primary return code            */
   AP_UINT32        secondary_rc;         /* secondary return code          */
   AP_UINT16        trace_flags;          /* trace flags                    */
   AP_UINT32        truncation_length;    /* truncate each msg to this size */
   unsigned char    init_flags;           /* TRUE if initializing flags     */
   unsigned char    set_flags;            /* TRUE if setting flags          */
                                          /*  FALSE if unsetting flags      */
   unsigned char    set_internal;         /* reserved                       */
   AP_UINT16        internal_level;       /* reserved                       */
} SET_TRACE_TYPE;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_SET_TRACE_TYPE

trace_flags

The types of tracing required. To turn off all tracing, or to turn on tracing of all types, specify one of the

following values:

AP_NO_TRACE

No tracing.

AP_ALL_TRACE

Tracing of all types.

To control tracing on specific interfaces, select one or more values from the list below, combined using a logical OR operation. For more information about these trace types, see “Trace Types”.

If *init_flags* is set to AP_YES, select the values corresponding to the interfaces where you want tracing to be active, and do not select the values corresponding to the interfaces where you want it to be inactive. If *init_flags* is set to AP_NO, select the values corresponding to the interfaces where you want to change the state of tracing.

AP_APPC_MSG	APPC messages
AP_FM_MSG	FM messages
AP_LUA_MSG	LUA messages
AP_NOF_MSG	NOF messages
AP_MS_MSG	MS messages
AP_PV_MSG	APPC and CPI-C messages for back-level computers
AP_DLPI_MSG	DLPI messages
AP_SDLC_MSG	SDLC messages (note that this option also provides additional detail in SDLC line tracing)
AP_NLI_MSG	NLI messages
AP_DLC_MSG	Node to DLC messages
AP_NODE_MSG	Node messages
AP_SLIM_MSG	Messages sent between master and backup servers
AP_DATAGRAM	Datagram messages

truncation_length

Specify the maximum length, in bytes, of the information to be written to the trace file for each message. This value must be at least 256.

If a trace message is longer than the length specified in this parameter, SNAplus2 writes only the start of the message to the trace file, and discards the data beyond *truncation_length*. This allows you to record the most important information for each message but avoid filling up the file with long messages.

To specify no truncation (all the data from each message is written to the file), set this parameter to zero.

init_flags

Specifies whether to initialize tracing (define the tracing state at all interfaces), or to change the state of tracing at one or more interfaces (leaving the others unchanged). Possible values are:

AP_YES

Tracing is being initialized. The *trace_flags* parameter defines the required state of tracing at all interfaces.

AP_NO

Tracing is being changed. The *trace_flags* parameter defines the interfaces where tracing is being activated or deactivated; other interfaces will not be affected.

set_flags

If *init_flags* is set to AP_NO, this parameter specifies whether tracing is to be activated or deactivated at the requested interfaces. Possible values are:

AP_YES

Tracing is to be activated at the interfaces specified by the *trace_flags* parameter.

AP_NO

Tracing is to be deactivated at the interfaces specified

by the *trace_flags* parameter.

If *init_flags* is set to AP_YES, this parameter is ignored.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

secondary_rc Not used.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_TRUNC_LEN

The *truncation_length* parameter specified a length of less than 256 bytes.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

Trace Types

Figure 5-1, “Overall Structure of SNAplus2,” shows the overall structure of SNAplus2. Each kernel-space trace type, relating to data transferred across a particular interface between SNAplus2 components, is shown in the diagram at the interface where it is traced.

Figure 5-1 Overall Structure of SNAplus2

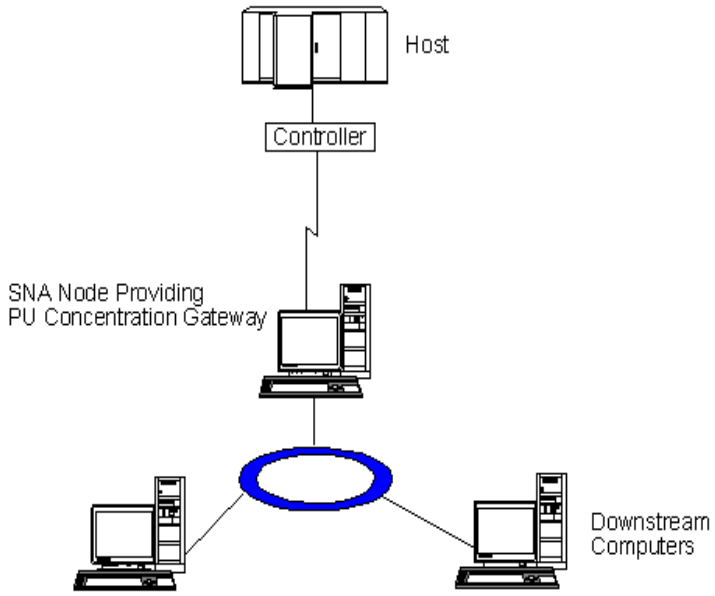


Figure 5-1, “Overall Structure of SNAplus2,” shows the following types of tracing, each of which can be controlled separately:

APPC messages

Messages between the APPC library and the node

FM messages

Messages between the 3270 emulation program and the node

LUA messages

Messages between the LUA library and the node

NOF messages

Messages between the NOF library and the node

MS messages

Messages between the MS library and the node

DLC line trace

SNA data sent on a DLC (tracing on these messages is controlled by the ADD_DLC_TRACE verb, not by SET_TRACE_TYPE)

Node to DLC messages

Messages between the APPN node and the DLC component

In addition, the following message types (internal to SNAplus2) can be traced:

Node messages

Messages between components within the APPN protocol code

SLIM messages

Messages sent between master and backup servers in a client-server system

Control messages

Internal control messages between system components

START_DLC

START_DLC requests the activation of a DLC.

This verb must be issued to a running node.

VCB Structure

```
typedef struct start_dlc
{
   AP_UINT16        opcode;                  /* verb operation code      */
   unsigned char    reserv2;                 /* reserved                 */
   unsigned char    format;                  /* reserved                 */
   AP_UINT16        primary_rc;              /* primary return code      */
   AP_UINT32        secondary_rc;            /* secondary return code    */
   unsigned char    dlc_name[8];             /* name of DLC              */
} START_DLC;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_START_DLC

dlc_name Name of the DLC to be started. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, which must match the name of a defined DLC.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameter:

primary_rc AP_OK

This return code indicates only that the verb was issued successfully; the verb does not wait for the DLC to initialize, and so does not return error return codes if the initialization of the DLC fails. DLC initialization failures are reported using messages written to the error log file.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_INVALID_DLC The *dlc_name* parameter was not the name of a defined DLC.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc	AP_STATE_CHECK
secondary_rc	AP_DLC_DEACTIVATING The specified DLC has already been started, and is in the process of being deactivated.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

START_INTERNAL_PU

START_INTERNAL_PU requests DLUR to initiate SSCP-PU session activation for a previously defined local PU which is served by DLUR.

This verb must be issued to a running node.

VCB Structure

```
typedef struct start_internal_pu
{
   AP_UINT16        opcode;                    /* verb operation code      */
   unsigned char    reserv2;                   /* reserved                 */
   unsigned char    format;                    /* reserved                 */
   AP_UINT16        primary_rc;                /* primary return code      */
   AP_UINT32        secondary_rc;              /* secondary return code    */
   unsigned char    pu_name[8];                /* internal PU name         */
   unsigned char    dlus_name[17];             /* DLUS name                */
   unsigned char    bkup_dlus_name[17];        /* Backup DLUS name         */
} START_INTERNAL_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_START_INTERNAL_PU

pu_name

Name of the internal PU to be started (which must have been previously defined using DEFINE_INTERNAL_PU). The name is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

dlus_name

Name of DLUS node which DLUR will contact to solicit SSCP-PU session activation for the given PU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8

A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To use the DLUS specified in the DEFINE_INTERNAL_PU verb, or the global default specified in DEFINE_DLUR_DEFAULTS if none was specified in DEFINE_INTERNAL_PU, set this parameter to 17 binary zeros.

bkup_dlus_name

Name of DLUS node which DLUR will store as the backup DLUS for the given PU. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

To use the backup DLUS specified in the DEFINE_INTERNAL_PU verb, or the global backup default specified in DEFINE_DLUR_DEFAULTS if none was specified in DEFINE_INTERNAL_PU, set this parameter to 17 binary zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DLUS_NAME

The *dlus_name* parameter contained a character that was not valid or was not in the correct format.

AP_INVALID_BKUP_DLUS_NAME

The *bkup_dlus_name* parameter contained a character that was not valid or was not in the correct format.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_NO_DEFAULT_DLUS_DEFINED

A DLUS name was not specified either on this verb or on the DEFINE_INTERNAL_PU verb, and there is no default DLUS defined because the DEFINE_DLUR_DEFAULTS verb has not been issued.

AP_PU_NOT_DEFINED

The supplied PU name was not the name of an internal PU defined using DEFINE_INTERNAL_PU.

AP_PU_ALREADY_ACTIVATING

The PU is already in the process of being started.

AP_PU_ALREADY_ACTIVE

The PU has already been started.

Returned Parameters: Unsuccessful

If the verb does not execute successfully, SNAplus2 returns the following parameters.

primary_rc AP_UNSUCCESSFUL

secondary_rc Possible values are:

AP_DLUS_REJECTED

The DLUS rejected the session activation request.

AP_DLUS_CAPS_MISMATCH

The configured DLUS name was not a DLUS node.

AP_PU_FAILED_ACTPU

The local node rejected a message from the DLUS. This may be caused by an internal error, a resource shortage, or a problem with the received message; check the SNAplus2 log files for messages providing more information.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The node does not support DLUR; this is defined by the *dlur_supported* parameter on DEFINE_NODE.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

START_LS

START_LS normally starts an inactive LS. Alternatively, it can be used to leave the LS inactive but specify that it can be activated automatically by SNAplus2 when required or activated by the remote system.

This verb must be issued to a running node.

VCB Structure

```
typedef struct start_ls
{
   AP_UINT16        opcode;          /* verb operation code                */
   unsigned char    reserv2;         /* reserved                           */
   unsigned char    format;          /* reserved                           */
   AP_UINT16        primary_rc;      /* primary return code                */
   AP_UINT32        secondary_rc;    /* secondary return code              */
   unsigned char    ls_name[8];      /* name of link station               */
   unsigned char    react_kicked;    /* retry in progress?                 */
   unsigned char    enable;          /* start ls or enable auto-activation? */
   unsigned char    reserv3[3];      /* reserved                           */
} START_LS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_START_LS

ls_name Name of the link station to be started. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, which must already have been defined by a DEFINE_LS verb.

enable Specifies the action to be taken for the LS.

To start the LS, set this parameter to AP_ACTIVATE.

To leave the LS inactive but specify that it can be activated (either by SNAplus2 or by the remote system) when required, specify one or both of the following values (combined using a logical OR):

AP_AUTO_ACT

The LS can be activated automatically by SNAplus2 when required for a session. This value should be used only when the LS is defined to be auto-activatable (*auto_act_supp* in the LS definition is set to AP_YES); it re-enables auto-activation after the LS has been manually stopped using STOP_LS.

AP_REMOTE_ACT

The LS can be activated by the remote system. This value does not alter the defined value of *disable_remote_act* in the LS definition; when the LS is next stopped, it will return to the defined setting.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_LINK_NAME_SPECIFIED

The *ls_name* parameter was not the name of a defined LS.

AP_INVALID_LINK_ENABLE

The *enable* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc	AP_STATE_CHECK
secondary_rc	Possible values are:
	AP_ACTIVATION_LIMITS_REACHED
	The activation limits have been reached.
	AP_LINK_DEACT_IN_PROGRESS
	The specified LS is currently being deactivated. You cannot start it until the deactivation process has finished.
	AP_ALREADY_STARTING
	The specified LS is already starting.
	AP_PARALLEL_TGS_NOT_SUPPORTED
	A link to the remote system is already active. The adjacent node does not support parallel transmission groups.
	AP_PORT_INACTIVE
	The LS cannot be started because its associated port is not active.
react_kicked	Specifies whether SNAplus2 will retry the attempt to activate the LS (based on the *react_timer_retry* parameter in the LS definition). Possible values are:
	AP_YES
	LS activation will be retried (up to the number of attempts specified by *react_timer_retry*).
	AP_NO
	LS activation will not be retried.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Cancelled

If the verb does not execute because it was cancelled by another verb, SNAplus2 returns the following parameters:

primary_rc AP_CANCELLED

secondary_rc Possible values are:

AP_NO_SECONDARY_RC

A STOP_LS verb was issued before the START_LS verb had completed. The START_LS verb was cancelled.

AP_LINK_DEACTIVATED

The DLC or port used by the LS was stopped before the START_LS verb had completed. The START_LS verb was cancelled.

react_kicked Specifies whether SNAplus2 will retry the attempt to activate the LS (based on the *react_timer_retry* parameter in the LS definition). Possible values are:

AP_YES

LS activation will be retried (up to the number of attempts specified by *react_timer_retry*).

AP_NO

LS activation will not be retried.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

START_PORT

START_PORT requests the activation of a port. The DLC specified for the port must be active before this verb is issued.

This verb must be issued to a running node.

VCB Structure

```
typedef struct start_port
{
   AP_UINT16        opcode;                  /* verb operation code      */
   unsigned char    reserv2;                 /* reserved                 */
   unsigned char    format;                  /* reserved                 */
   AP_UINT16        primary_rc;              /* primary return code      */
   AP_UINT32        secondary_rc;            /* secondary return code    */
   unsigned char    port_name[8];            /* name of port             */
} START_PORT;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_START_PORT

port_name Name of port to be started. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, which must already have been defined by a DEFINE_PORT verb.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_PORT

The *port_name* parameter was not the name of a defined port.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_DLC_INACTIVE

The port cannot be started because its associated DLC is not active.

AP_DUPLICATE_PORT

The specified port has already been started.

AP_STOP_PORT_PENDING

The specified port is currently being deactivated. You cannot start it until the deactivation process has finished.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Cancelled

If the verb does not execute because it was cancelled, SNAplus2 returns the following parameters.

primary_rc AP_CANCELLED

A STOP_PORT verb was issued before this verb had completed. The START_PORT verb was cancelled.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

STOP_DLC

STOP_DLC requests SNAplus2 to stop a DLC; this also stops any active ports and LSs that use the DLC.

This verb must be issued to a running node.

VCB Structure

```
typedef struct stop_dlc
{
   AP_UINT16        opcode;                  /* verb operation code      */
   unsigned char    reserv2;                 /* reserved                 */
   unsigned char    format;                  /* reserved                 */
   AP_UINT16        primary_rc;              /* primary return code      */
   AP_UINT32        secondary_rc;            /* secondary return code    */
   unsigned char    stop_type;               /* stop type                */
   unsigned char    dlc_name[8];             /* name of DLC              */
} STOP_DLC;
```

Supplied Parameters

The application supplies the following parameters:

opcode — AP_STOP_DLC

stop_type — Type of stop processing required. Possible values are:

AP_ORDERLY_STOP

SNAplus2 will perform cleanup operations before stopping the DLC.

AP_IMMEDIATE_STOP

SNAplus2 will stop the DLC immediately.

dlc_name — Name of DLC to be stopped. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, which must already have been defined by a DEFINE_DLC verb.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_DLC

The *dlc_name* parameter did not match the name of a defined DLC.

AP_UNRECOGNIZED_DEACT_TYPE

The *stop_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_STOP_DLC_PENDING

The specified DLC is already in the process of being stopped.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Cancelled

If the verb does not execute because it has been cancelled, SNAplus2

returns the following parameters:

primary_rc AP_CANCELLED

The *stop_type* parameter specified an orderly stop, but the DLC was then stopped by a second command specifying an immediate stop or by a failure condition.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

STOP_INTERNAL_PU

STOP_INTERNAL_PU requests DLUR to initiate SSCP-PU session deactivation for a previously defined local PU which is served by DLUR.

This verb must be issued to a running node.

VCB Structure

```
typedef struct stop_internal_pu
{
   AP_UINT16          opcode                /* verb operation code      */
   unsigned char      reserv2;              /* reserved                 */
   unsigned char      format;               /* reserved                 */
   AP_UINT16          primary_rc;           /* primary return code
   AP_UINT32          secondary_rc;         /* secondary return code    */
   unsigned char      pu_name[8];           /* internal PU name         */
   unsigned char      stop_type;            /* type of stop requested   */
} STOP_INTERNAL_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_STOP_INTERNAL_PU

pu_name Name of the internal PU for which the SSCP-PU session will be deactivated. This is an 8-byte type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

stop_type Specifies how to stop the PU. Possible values are:

AP_ORDERLY_STOP

Deactivate all underlying PLU-SLU and SSCP-LU sessions before deactivating the SSCP-PU session.

AP_IMMEDIATE_STOP

Deactivate the SSCP-PU session immediately.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_STOP_TYPE

The *stop_type* parameter was not set to a valid value.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc Possible values are:

AP_PU_NOT_DEFINED

The supplied PU name did not match the name of a defined internal PU.

AP_PU_ALREADY_DEACTIVATING

The PU is already in the process of being stopped.

AP_PU_NOT_ACTIVE

The PU is not active.

Returned Parameters: Function Not Supported

If the verb does not execute because the node's configuration does not support it, SNAplus2 returns the following parameter:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The node does not support DLUR; this is defined by the *dlur_supported* parameter on DEFINE_NODE.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

STOP_LS

STOP_LS stops an active LS. Alternatively, it can be issued for an inactive LS, to specify that the LS cannot be activated automatically by SNAplus2 when required or activated by the remote system; if both of these are disabled, the LS can be activated only by issuing START_LS.

This verb must be issued to a running node.

VCB Structure

```
typedef struct stop_ls
{
   AP_UINT16        opcode;           /* verb operation code               */
   unsigned char    reserv2;          /* reserved                          */
   unsigned char    format;           /* reserved                          */
   AP_UINT16        primary_rc;       /* primary return code               */
   AP_UINT32        secondary_rc;     /* secondary return code             */
   unsigned char    stop_type;        /* stop type                         */
   unsigned char    ls_name[8];       /* name of link station              */
   unsigned char    disable;          /* disable remote or auto activation? */
   unsigned char    reserved[3]       /* reserved                          */
} STOP_LS;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_STOP_LS

stop_type Type of stop processing required. Possible values are:

AP_ORDERLY_STOP

SNAplus2 will perform cleanup operations before stopping the LS.

AP_IMMEDIATE_STOP

SNAplus2 will stop the LS immediately.

ls_name Name of LS to be stopped. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes, which must already have been

defined by a DEFINE_LS verb.

disable Specifies the action to be taken for the LS.

To stop an active LS and return to the default settings for auto-activation and remote activation, set this parameter to AP_NO.

To specify that an inactive LS cannot be activated by SNAplus2, or cannot be activated by the remote system, specify one or both of the following values (combined using a logical OR):

AP_AUTO_ACT

The LS cannot be activated automatically by SNAplus2.

AP_REMOTE_ACT

The LS cannot be activated by the remote system. This value does not alter the defined value of *disable_remote_act* in the LS definition; when the LS is next started and stopped, it will return to the defined setting.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_LINK_DEACT_IN_PROGRESS

The specified LS is already in the process of being deactivated.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_LINK_NOT_DEFD

The *ls_name* parameter did not match the name of a defined LS.

AP_UNRECOGNIZED_DEACT_TYPE

The *stop_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Cancelled

If the verb does not execute because it was cancelled, SNAplus2 returns the following parameters.

primary_rc AP_CANCELLED

The *stop_type* parameter specified an orderly stop, but the LS was then stopped by a second verb specifying an immediate stop or by a failure condition.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

STOP_PORT

STOP_PORT allows the application to stop a port. This also stops any active LSs that are using the port.

This verb must be issued to a running node.

VCB Structure

```
typedef struct stop_port
{
   AP_UINT16        opcode;                  /* verb operation code      */
   unsigned char    reserv2;                 /* reserved                 */
   unsigned char    format;                  /* reserved                 */
   AP_UINT16        primary_rc;              /* primary return code      */
   AP_UINT32        secondary_rc;            /* secondary return code    */
   unsigned char    stop_type;               /* Stop Type                */
   unsigned char    port_name[8];            /* name of port             */
} STOP_PORT;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_STOP_PORT

stop_type Type of stop processing required. Possible values are:

AP_ORDERLY_STOP

SNAplus2 will perform cleanup operations before stopping the port.

AP_IMMEDIATE_STOP

SNAplus2 will stop the port immediately.

port_name Name of port to be stopped. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following

parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc Possible values are:

AP_INVALID_PORT_NAME

The *port_name* parameter did not match the name of a defined port.

AP_UNRECOGNIZED_DEACT_TYPE

The *stop_type* parameter was not set to a valid value.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: State Check

If the verb does not execute because of a state error, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc AP_STOP_PORT_PENDING

The specified port is already in the process of being deactivated.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_STATE_CHECK, which are common to all NOF verbs.

Returned Parameters: Cancelled

If the verb does not execute because it has been cancelled, SNAplus2 returns the following parameters:

primary_rc AP_CANCELLED

The *stop_type* parameter specified an orderly stop, but the port was then stopped by a second verb specifying an immediate stop or by a failure condition.

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

TERM_NODE

TERM_NODE allows the application to stop the node with a specified urgency. This also stops all connectivity resources associated with the node.

If an RJE workstation is using LUs owned by the node, it will not automatically stop when the node is stopped (unless it was started with the "stop after inactivity" option); it must be stopped manually.

This verb must be issued to a running node.

VCB Structure

```
typedef struct term_node
{
   AP_UINT16        opcode;                  /* verb operation code      */
   unsigned char    reserv2;                 /* reserved                 */
   unsigned char    format;                  /* reserved                 */
   AP_UINT16        primary_rc;              /* primary return code      */
   AP_UINT32        secondary_rc;            /* secondary return code    */
   unsigned char    stop_type;               /* stop type                */
} TERM_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_TERM_NODE

stop_type Specifies how SNAplus2 should stop the node. Possible values are:

AP_ABORT

Stop immediately without attempting any cleanup processing. This value should be used only in serious error conditions, because it may cause problems for other programs using the node's resources.

AP_SHUTDOWN

Deactivate all LSs associated with the node before stopping.

AP_QUIESCE

Indicate to the network that the node is quiesced, reset session limits on all modes, unbind all endpoint sessions for the node's LUs, and then stop as for AP_SHUTDOWN.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of primary and secondary return codes that are common to all NOF verbs.

UNREGISTER_INDICATION_SINK

UNREGISTER_INDICATION_SINK unregisters the NOF application so that it no longer receives indications of a particular type (previously specified using REGISTER_INDICATION_SINK).

If the application has registered more than once to accept multiple indication types, it must unregister separately for each indication that it no longer wants to receive.

This verb must always be issued using the asynchronous NOF API entry point, including the callback routine that was supplied on the REGISTER_INDICATION_SINK verb. (For more information about the asynchronous NOF API entry point, see "Asynchronous Entry Point: nof_async").

This verb may be issued to the domain configuration file, to a running node or to a server where the node is not running, or to the `sna.net` file, depending on the type of indication for which the application is unregistering.

VCB Structure

```
typedef struct unregister_indication_sink
{
   AP_UINT16        opcode;             /* verb operation code              */
   unsigned char    reserv2;            /* reserved                         */
   unsigned char    format;             /* reserved                         */
   AP_UINT16        primary_rc;         /* primary return code              */
   AP_UINT32        secondary_rc;       /* secondary return code            */
   AP_UINT32        proc_id;            /* reserved                         */
   AP_UINT16        queue_id;           /* reserved                         */
   AP_UINT16        indication_opcode;  /* opcode of indication to be unsunk */
} UNREGISTER_INDICATION_SINK;
```

Supplied Parameters

The application supplies the following parameters:

opcode

AP_UNREGISTER_INDICATION_SINK

indication_opcode

The *opcode* parameter of the indication that is no longer required.

Returned Parameters: Successful Execution

If the verb executes successfully, SNAplus2 returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_OP_CODE

The *indication_opcode* parameter did not match the *opcode* of any of the SNAplus2 NOF indications, or the application has not previously registered to receive the specified indication on this target handle.

Appendix A, “Common Return Codes,” lists further secondary return codes associated with AP_PARAMETER_CHECK, which are common to all NOF verbs.

Returned Parameters: Function Not Supported

If the verb does not execute successfully because the local node does not support the function associated with the specified indication, SNAplus2 returns the following parameters:

primary_rc AP_FUNCTION_NOT_SUPPORTED

The local node does not support the specified indication. For details of the support required for each indication, see the description of each indication in Chapter 6, “NOF Indications.”

Returned Parameters: Other Conditions

Appendix A, “Common Return Codes,” lists further combinations of

primary and secondary return codes that are common to all NOF verbs.

6 NOF Indications

Overview

This chapter provides the following information for each NOF indication:

- Description of the indication's purpose and usage
- Verb control block (VCB) structure, as defined in the NOF API header file `nof_c.h`
- Explanations of the parameters returned to the application in the VCB

For information about how the application registers to receive NOF indications, see “REGISTER_INDICATION_SINK”.

CONFIG_INDICATION

This indication is generated when another NOF application or a SNAplus2 administration tool makes a change to the target configuration, when the target node is stopped or started, or when a DLC, port, or LS owned by the target node is stopped or started. The target can be the domain configuration file, a running node, or an inactive node on a server where the SNAplus2 software is running. The target is identified by the *target_handle* parameter on the REGISTER_INDICATION_SINK verb that registers to receive this indication.

VCB Structure

No specific VCB structure is associated with this indication. To register for configuration indications, the application specifies the value AP_CONFIG_INDICATION as the *indication_opcode* parameter on REGISTER_INDICATION_SINK. When a change is made, SNAplus2 then reports this to the application's callback routine by sending a copy of the VCB from the NOF verb that made the change. For example, if the configuration was modified by a DEFINE_DLC verb, SNAplus2 sends a copy of the DEFINE_DLC VCB to the application's callback routine.

To enable the application to distinguish between configuration indications and asynchronous responses to its own NOF verbs, SNAplus2 changes the *primary_rc* parameter in the VCB for a configuration indication. The value AP_INDICATION identifies a VCB associated with a configuration indication; the value AP_OK, or any other value, indicates an asynchronous response to one of the application's own NOF verbs.

The following events are not reported as configuration indications:

- Changes to the SNA network file sna.net. To receive indications of these changes, the application must register for the indication type AP_SNA_NET_INDICATION. For more information, see “SNA_NET_INDICATION”.
- Starting and stopping the SNA software on other servers. To receive indications of these changes, the application must register for the indication type AP_SERVER_INDICATION. For more information, see “SERVER_INDICATION”.

The range of VCBs that can be returned as configuration indications depends on the type of target handle specified on REGISTER_INDICATION_SINK:

Domain configuration file

The application can receive VCBs for any verbs that modify domain resources but not node resources (verbs that can be issued to the domain configuration file).

Node configuration file

The application can receive VCBs for any verbs that modify node resources.

Running node

The application can receive VCBs for any verbs that modify node resources, TERM_NODE VCBs, and START_* and STOP_* VCBs for DLCs, ports, and LSs.

Inactive node

The application can receive VCBs for any verbs that modify node resources and also INIT_NODE VCBs.

DIRECTORY_INDICATION

This indication is generated when an entry is added to or removed from the local directory database.

VCB Structure

```
typedef struct directory_indication
{
   AP_UINT16        opcode;              /* verb operation code              */
   unsigned char    reserv2;             /* reserved                         */
   unsigned char    format;              /* reserved                         */
   AP_UINT16        primary_rc;          /* primary return code              */
   AP_UINT32        secondary_rc;        /* secondary return code            */
   unsigned char    data_lost;           /* previous indication lost         */
   unsigned char    removed;             /* is entry being removed?          */
   unsigned char    resource_name[17];   /* resource name                    */
   unsigned char    invalid;            /* reserved                         */
   AP_UINT16        resource_type;       /* resource type                    */
   unsigned char    parent_name[17];     /* parent resource name             */
   unsigned char    entry_type;          /* type of the directory entry      */
   AP_UINT16        parent_type;         /* parent resource type             */
   unsigned char    description[32];     /* resource description             */
   unsigned char    reserv3[16];         /* reserved                         */
   unsigned char    reserva[20];         /* reserved                         */
} DIRECTORY_INDICATION;
```

Parameters

opcode AP_DIRECTORY_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous directory indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous directory indications were lost.

Later fields in this VCB may be set to zeros.

AP_NO

No previous directory indications were lost.

removed Specifies whether the indicated resource has been removed from the directory or added to it. Possible values are:

AP_YES

The entry has been removed.

AP_NO

The entry has been added.

resource_name Fully qualified name of the resource. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

resource_type Resource type. Possible values are:

AP_NNCP_RESOURCE

Network node.

AP_ENCP_RESOURCE

End node.

AP_LU_RESOURCE

LU.

parent_name Fully qualified name of parent resource. If *resource_type* is AP_NNCP_RESOURCE, this is set to 17 binary zeros.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

entry_type Specifies the type of the directory entry. This is one of the following:

AP_HOME

Local resource.

AP_CACHE

Cached entry.

parent_type Specifies the parent type of the resource being registered. If *resource_type* is AP_NNCP_RESOURCE, this parameter is not used. Possible values are:

AP_NNCP_RESOURCE

Network node.

AP_ENCP_RESOURCE

End node.

description A null-terminated text string describing the resource, as specified in the definition of the resource.

DLC_INDICATION

This indication is generated when a DLC changes state between active and inactive.

VCB Structure

```
typedef struct dlc_indication
{
   AP_UINT16        opcode;             /* verb operation code             */
   unsigned char    reserv2;            /* reserved                        */
   unsigned char    format;             /* reserved                        */
   AP_UINT16        primary_rc;         /* primary return code             */
   AP_UINT32        secondary_rc;       /* secondary return code           */
   unsigned char    data_lost;          /* previous indication lost        */
   unsigned char    deactivated;        /* has session been deactivated?   */
   unsigned char    dlc_name[8];        /* link station name               */
   unsigned char    description[32];    /* resource description            */
   unsigned char    reserv1[16];        /* reserved                        */
   unsigned char    reserva[20];        /* reserved                        */
} DLC_INDICATION;
```

Parameters

opcode AP_DLC_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous DLC indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous DLC indications were lost.

deactivated Specifies whether the DLC has become inactive or become active. Possible values are:

AP_YES

The DLC has become inactive.

AP_NO

The DLC has become active.

dlc_name Name of DLC. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

description A null-terminated text string describing the DLC, as specified in the definition of the DLC.

DLUR_LU_INDICATION

This indication is generated when a DLUR LU is activated or deactivated. This indication can be used by a registered application to maintain a list of currently active DLUR LUs.

VCB Structure

```
typedef struct dlur_lu_indication
{
   AP_UINT16        opcode;             /* Indication operation code        */
   unsigned char    reserv2;            /* reserved                         */
   unsigned char    format;             /* reserved                         */
   AP_UINT16        primary_rc;         /* primary return code              */
   AP_UINT32        secondary_rc;       /* secondary return code            */
   unsigned char    data_lost;          /* Previous indication lost?        */
   unsigned char    reason;             /* reason for this indication       */
   unsigned char    lu_name[8];         /* LU name                          */
   unsigned char    pu_name;            /* PU name                          */
   unsigned char    nau_address;        /* NAU address                      */
   unsigned char    reserv5[7];         /* reserved                         */
 } DLUR_LU_INDICATION;
```

Parameters

opcode AP_DLUR_LU_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous directory indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous directory indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

	No previous directory indications were lost.
reason	Reason for this indication. Possible values are: AP_ADDED The DLUR has just been activated by the DLUS. AP_REMOVED The DLUR has been deactivated, either explicitly by the DLUS or implicitly by a link failure or the deactivation of the PU.
lu_name	Name of the logical unit (LU). This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.
pu_name	Name of the physical unit (PU) that this LU uses. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.
nau_address	Network accessible unit (NAU) address of the LU. This value must be in the range 1-255.

DLUR_PU_INDICATION

This indication is generated when a physical unit (PU) for the node that supports the dependent LU requester (DLUR) function is attempting to activate, fails to activate, activates, or is deactivated. This indication can be used to maintain a list of currently active DLUR PUs.

VCB Structure

```
typedef struct dlur_pu_indication
{
   AP_UINT16        opcode;              /* Indication operation code        */
   unsigned char    reserv2;             /* reserved                         */
   unsigned char    format;              /* reserved                         */
   AP_UINT16        primary_rc;          /* primary return code              */
   AP_UINT32        secondary_rc;        /* secondary return code            */
   unsigned char    data_lost;           /* Previous indication lost?        */
   unsigned char    reason;              /* reason for this indication       */
   unsigned char    pu_name[8];          /* PU name                          */
   unsigned char    pu_id[4];            /* PU identifier                    */
   unsigned char    pu_location;         /* downstream or local PU           */
   unsigned char    pu_status;           /* status of the PU                 */
   unsigned char    dlus_name[17];       /* current DLUS name                */
   unsigned char    dlus_session_status; /* status of the DLUS pipe          */
   unsigned char    reserv5[2];          /* reserved                         */
} DLUR_PU_INDICATION;
```

Parameters

opcode

AP_DLUR_PU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous directory indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting

the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous directory indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous directory indications were lost.

reason

Cause of the indication. Possible values are:

Possible values are:

AP_ACTIVATION_STARTED

The PU is activating.

AP_ACTIVATING

The PU has become active.

AP_DEACTIVATING

The PU has become inactive.

AP_FAILED

The PU has failed.

AP_ACTIVATION_FAILED

The PU has failed to activate.

pu_name

Name of the physical unit (PU). This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

pu_id

PU identifier defined in a DEFINE_INTERNAL_PU verb or obtained in an XID from a downstream PU. This value is a 4-byte hexadecimal string. Bits 0-11 are set to the Block number and bits 12-31 are set to the ID number that uniquely identifies the PU.

pu_location

Location of the PU. Possible values are:

AP_INTERNAL

The PU has been defined by a DEFINE_INTERNAL_PU verb.

AP_DOWNSTREAM

The PU is located at a downstream computer.

dlur_pu_detail.pu_status

Status of the PU, as seen by the DLUR. Possible values are:

AP_RESET_NO_RETRY

The PU is in reset state and will not be retried.

AP_RESET_RETRY

The PU is in reset state and will be retried.

AP_PEND_ACTPU

The PU is waiting for an ACTPU from the host.

AP_PEND_ACTPU_RSP

Having forwarded an ACTPU to the PU, DLUR is now waiting for the PU to respond to it.

AP_ACTIVE

The PU is active.

AP_PEND_DACTPU_RSP

Having forwarded a DACTPU to the PU, DLUR is now waiting for the PU to respond to it.

AP_PEND_INOP

DLUR is waiting for all necessary events to complete before it deactivates the PU.

dlus_name

Name of the dependent LU server (DLUS) node that the PU is currently using (or attempting to use). The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up

to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters. If the *reason* parameter is set to AP_FAILED, the *dlus_name* parameter is set to all zeros.

dlur_pu_detail.dlus_session_status

Status of the DLUS pipe currently being used by the PU. Possible values are:

AP_PENDING_ACTIVE

The DLUS pipe is currently being activated.

AP_ACTIVE

The DLUS pipe is active.

AP_PENDING_INACTIVE

The DLUS pipe is currently being deactivated.

AP_INACTIVE

The DLUS pipe is inactive.

DLUS_INDICATION

This indication is generated when a pipe to a DLUS node changes state between active and inactive. When the pipe becomes inactive, the indication also includes pipe statistics.

VCB Structure

```
typedef struct dlus_indication
{
   AP_UINT16        opcode;             /* verb operation code             */
   unsigned char    reserv2[2];         /* reserved                        */
   AP_UINT16        primary_rc;         /* primary return code             */
   AP_UINT32        secondary_rc;       /* secondary return code           */
   unsigned char    data_lost;          /* previous indication lost        */
   unsigned char    deactivated;        /* has DLUS become inactive?       */
   unsigned char    dlus_name[17];      /* DLUS name                       */
   unsigned char    reserv1;            /* reserved                        */
   PIPE_STATS       pipe_stats;         /* pipe statistics                 */
   unsigned char    reserva[20];        /* reserved                        */
} DLUS_INDICATION;


typedef struct pipe_stats
{
   AP_UINT32       reqactpu_sent;           /* REQACTPUs sent to DLUS       */
   AP_UINT32       reqactpu_rsp_received;   /* RSP(REQACTPU)s received      */
                                            /*  from DLUS                   */
   AP_UINT32       actpu_received;          /* ACTPUs received from DLUS    */
   AP_UINT32       actpu_rsp_sent;          /* RSP(ACTPU)s sent to DLUS     */
   AP_UINT32       reqactpu_sent;           /* REQDACTPUs sent to DLUS      */
   AP_UINT32       reqdactpu_rsp_received;  /* RSP(REQDACTPU)s received     */
                                            /*  from DLUS                   */
   AP_UINT32       dactpu_received;         /* DACTPUs received from DLUS   */
   AP_UINT32       dactpu_rsp_sent;         /* RSP(DACTPU)s sent to DLUS    */
   AP_UINT32       actlu_received;          /* ACTLUs received from DLUS    */
   AP_UINT32       actlu_rsp_sent;          /* RSP(ACTLU)s sent to DLUS     */
   AP_UINT32       dactlu_received;         /* DACTLUs received from DLUS   */
   AP_UINT32       dactlu_rsp_sent;         /* RSP(DACTLU)s sent to DLUS    */
   AP_UINT32       sscp_pu_mus_rcvd;        /* MUs for SSCP-PU sessions rcvd */
   AP_UINT32       sscp_pu_mus_sent;        /* MUs for SSCP-PU sessions sent */
   AP_UINT32       sscp_lu_mus_rcvd;        /* MUs for SSCP-LU sessions rcvd */
   AP_UINT32       sscp_lu_mus_sent;        /* MUs for SSCP-LU sessions sent */
} PIPE_STATS;
```

Parameters

opcode AP_DLUS_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous DLUS indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous downstream LU indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous downstream LU indications were lost.

deactivated Specifies whether the pipe has become inactive or become active. Possible values are:

AP_YES

The pipe has become inactive.

AP_NO

The pipe has become active.

dlus_name Name of the DLUS node. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

If the pipe was deactivated, a pipe_stats structure is included. The fields in this structure are as follows:

pipe_stats.reqactpu_sent

Number of REQACTPUs sent to DLUS over the pipe.

pipe_stats.reqactpu_rsp_received

Number of RSP(REQACTPU)s received from DLUS

over the pipe.

pipe_stats.actpu_received

Number of ACTPUs received from DLUS over the pipe.

pipe_stats.actpu_rsp_sent

Number of RSP(ACTPU)s sent to DLUS over the pipe.

pipe_stats.reqdactpu_sent

Number of REQDACTPUs sent to DLUS over the pipe.

pipe_stats.reqdactpu_rsp_received

Number of RSP(REQDACTPU)s received from DLUS over the pipe.

pipe_stats.dactpu_received

Number of DACTPUs received from DLUS over the pipe.

pipe_stats.dactpu_rsp_sent

Number of RSP(DACTPU)s sent to DLUS over the pipe.

pipe_stats.actlu_received

Number of ACTLUs received from DLUS over the pipe.

pipe_stats.actlu_rsp_sent

Number of RSP(ACTLU)s sent to DLUS over the pipe.

pipe_stats.dactlu_received

Number of DACTLUs received from DLUS over the pipe.

pipe_stats.dactlu_rsp_sent

Number of RSP(DACTLU)s sent to DLUS over the pipe.

pipe_stats.sscp_pu_mus_rcvd

Number of SSCP-PU MUs received from DLUS over the pipe.

pipe_stats.sscp_pu_mus_sent

Number of SSCP-PU MUs sent to DLUS over the pipe.

pipe_stats.sscp_lu_mus_rcvd

Number of SSCP-LU MUs received from DLUS over the pipe.

pipe_stats.sscp_lu_mus_sent

Number of SSCP-LU MUs sent to DLUS over the pipe.

DOWNSTREAM_LU_INDICATION

This indication is generated when either the LU-SSCP session or the PLU-SLU session between the downstream LU and the host changes state between active and inactive. When one of these sessions becomes inactive, the indication also includes session statistics for that session.

VCB Structure

```
typedef struct downstream_lu_indication
{
   unsigned short  opcode;              /* verb operation code             */
   unsigned char   reserv2;             /* reserved                        */
   unsigned char   format;              /* reserved                        */
   unsigned short  primary_rc;          /* primary return code             */
   unsigned long   secondary_rc;        /* secondary return code           */
   unsigned char   data_lost;           /* previous indication lost        */
   unsigned char   dspu_name[8];        /* PU name                         */
   unsigned char   ls_name[8];          /* Link station name               */
   unsigned char   dslu_name[8];        /* LU name                         */
   unsigned char   description[32];     /* resource description            */
   unsigned char   reserv3[16];         /* reserved                        */
   unsigned char   nau_address;         /* NAU address                     */
   unsigned char   lu_sscp_sess_active; /* Is LU-SSCP session active       */
   unsigned char   plu_sess_active;     /* Is PLU-SLU session active       */
   unsigned char   dspu_services;       /* DSPU services                   */
   unsigned char   reserv1;             /* reserved                        */
   SESSION_STATS   lu_sscp_stats;       /* LU-SSCP session statistics      */
   SESSION_STATS   ds_plu_stats;        /* Downstream PLU-SLU session stats */
   SESSION_STATS   us_plu_stats;        /* Upstream PLU-SLU session stats  */
} DOWNSTREAM_LU_INDICATION;


typedef struct session_stats
{
  unsigned short  rcv_ru_size;          /* session receive RU size          */
  unsigned short  send_ru_size;         /* session send RU size             */
  unsigned short  max_send_btu_size;    /* maximum send BTU size            */
  unsigned short  max_rcv_btu_size;     /* maximum rcv BTU size             */
  unsigned short  max_send_pac_win;     /* maximum send pacing window size  */
  unsigned short  cur_send_pac_win;     /* current send pacing window size  */
  unsigned short  max_rcv_pac_win;      /* maximum receive pacing window size*/
  unsigned short  cur_rcv_pac_win;      /* current receive pacing window size*/
```

```
  unsigned long   send_data_frames;     /* number of data frames sent        */
  unsigned long   send_fmd_data_frames; /* num fmd data frames sent          */
  unsigned long   send_data_bytes;      /* number of data bytes sent         */
  unsigned long   rcv_data_frames;      /* number of data frames received    */
  unsigned long   rcv_fmd_data_frames;  /* num fmd data frames received      */
  unsigned long   rcv_data_bytes;       /* number of data bytes received     */
  unsigned char   sidh;                 /* session ID high byte (from LFSID) */
  unsigned char   sidl;                 /* session ID low byte (from LFSID)  */
  unsigned char   odai;                 /* ODAI bit set                      */
  unsigned char   ls_name[8];           /* Link station name                 */
  unsigned char   reserve;              /* reserved                          */
} SESSION_STATS;
```

Parameters

opcode

AP_DOWNSTREAM_LU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous downstream LU indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous downstream LU indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous downstream LU indications were lost.

dspu_name

Name of the downstream PU associated with the downstream LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

ls_name

Name of the link station associated with the downstream LU. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

dslu_name

Name of the downstream LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

description

A null-terminated text string describing the downstream LU, as specified in the definition of the LU.

nau_address

Network accessible unit address of the LU.

lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is not active.

plu_sess_active

Specifies whether the PLU-SLU session is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is not active.

dspu_services

Specifies the services provided by the local node to the downstream LU.

Possible values are:

AP_PU_CONCENTRATION

Downstream LU is served by PU concentration.

AP_DLUR

Downstream LU is served by DLUR.

If the LU-SSCP session was deactivated, a session_stats structure is included for this session; if the PLU-SLU session was deactivated, session_stats structures are included for the downstream and upstream PLU-SLU sessions. The fields in this structure are as follows:

rcv_ru_size

Maximum receive RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

send_ru_size

Maximum send RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

max_send_btu_size

Maximum BTU size that can be sent.

max_rcv_btu_size

Maximum BTU size that can be received.

max_send_pac_win

Maximum size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_send_pac_win

Current size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

max_rcv_pac_win

Maximum size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_rcv_pac_win

Current size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

send_data_frames

Number of normal flow data frames sent.

send_fmd_data_frames

Number of normal flow FMD data frames sent.

send_data_bytes

Number of normal flow data bytes sent.

rcv_data_frames

Number of normal flow data frames received.

rcv_fmd_data_frames

Number of normal flow FMD data frames received.

rcv_data_bytes

Number of normal flow data bytes received.

sidh

Session ID high byte. (In the upstream PLU-SLU session statistics, this parameter is reserved.)

sidl

Session ID low byte. (In the upstream PLU-SLU session statistics, this parameter is reserved.)

odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station. (In the upstream PLU-SLU session statistics, this parameter is reserved.)

ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with

spaces if the name is shorter than 8 characters. (In the upstream PLU-SLU session statistics, this parameter is reserved if *dspu_services* is set to AP_PU_CONCENTRATION.)

DOWNSTREAM_PU_INDICATION

This indication is generated when the PU-SSCP session between the downstream PU and the host changes state between active and inactive. When the session becomes inactive, the indication also includes PU-SSCP session statistics.

VCB Structure

```
typedef struct downstream_pu_indication
{
 unsigned short    opcode;                 /* verb operation code           */
 unsigned char     reserv2[2];             /* reserved                      */
 unsigned short    primary_rc;             /* primary return code           */
 unsigned long     secondary_rc;           /* secondary return code         */
 unsigned char     data_lost;              /* previous indication lost      */
 unsigned char     dspu_name[8];           /* PU name                       */
 unsigned char     description[32];        /* resource description          */
 unsigned char     reserv3[16];            /* reserved                      */
 unsigned char     ls_name[8];             /* Link station name             */
 unsigned char     pu_sscp_sess_active;    /* Is PU-SSCP session active     */
 unsigned char     dspu_services;          /* DSPU services                 */
 unsigned char     reserv1[2];             /* reserved                      */
 SESSION_STATS     pu_sscp_stats;          /* PU-SSCP session statistics    */
} DOWNSTREAM_PU_INDICATION;


typedef struct session_stats
{
 unsigned short   rcv_ru_size;          /* session receive RU size          */
 unsigned short   send_ru_size;         /* session send RU size             */
 unsigned short   max_send_btu_size;    /* maximum send BTU size            */
 unsigned short   max_rcv_btu_size;     /* maximum rcv BTU size             */
 unsigned short   max_send_pac_win;     /* maximum send pacing window size  */
 unsigned short   cur_send_pac_win;     /* current send pacing window size  */
 unsigned short   max_rcv_pac_win;      /* maximum receive pacing window size*/
 unsigned short   cur_rcv_pac_win;      /* current receive pacing window size*/
 unsigned long    send_data_frames;     /* number of data frames sent       */
 unsigned long    send_fmd_data_frames; /* num fmd data frames sent         */
 unsigned long    send_data_bytes;      /* number of data bytes sent        */
 unsigned long    rcv_data_frames;      /* number of data frames received   */
 unsigned long    rcv_fmd_data_frames;  /* num fmd data frames received     */
 unsigned long    rcv_data_bytes;       /* number of data bytes received    */
```

```
 unsigned char    sidh;               /* session ID high byte (from LFSID) */
 unsigned char    sidl;               /* session ID low byte (from LFSID)  */
 unsigned char    odai;               /* ODAI bit set                      */
 unsigned char    ls_name[8];         /* Link station name                 */
 unsigned char    reserve;            /* reserved                          */
} SESSION_STATS;
```

Parameters

opcode

AP_DOWNSTREAM_PU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous downstream PU indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous downstream PU indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous downstream PU indications were lost.

dspu_name

Name of downstream PU. The name is an 8-byte EBCDIC type-A string, padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

description

A null-terminated text string describing the downstream PU, as specified in the definition of the LS associated with the PU.

ls_name

Name of the link station associated with the downstream PU. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 characters.

pu_sscp_sess_active

Specifies whether the PU-SSCP session to the downstream PU is active. Possible values are:

AP_YES

The session is active.

AP_NO

The session is not active.

dspu_services

Specifies the services provided by the local node to the downstream PU.

Possible values are:

AP_PU_CONCENTRATION

Downstream LU is served by PU concentration.

AP_DLUR

Downstream LU is served by DLUR.

pu_sscp_stats.rcv_ru_size

Reserved (always set to zero).

pu_sscp_stats.send_ru_size

Reserved (always set to zero).

pu_sscp_stats.max_send_btu_size

Maximum BTU size that can be sent.

pu_sscp_stats.max_rcv_btu_size

Maximum BTU size that can be received.

pu_sscp_stats.max_send_pac_win

Reserved (always set to zero).

pu_sscp_stats.cur_send_pac_win

Reserved (always set to zero).

pu_sscp_stats.max_rcv_pac_win

Reserved (always set to zero).

pu_sscp_stats.cur_rcv_pac_win

Reserved (always set to zero).

pu_sscp_stats.send_data_frames

Number of normal flow data frames sent.

pu_sscp_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

pu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

pu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

pu_sscp_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

pu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

pu_sscp_stats.sidh

Session ID high byte.

pu_sscp_stats.sidl

Session ID low byte.

pu_sscp_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station.

pu_sscp_stats.ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with

spaces if the name is shorter than 8 characters.

FOCAL_POINT_INDICATION

This indication is generated whenever a focal point is acquired, changed or revoked.

VCB Structure

```
typedef struct focal_point_indication
{
  AP_UINT16        opcode;             /* verb operation code               */
  unsigned char    reserv2;            /* reserved                          */
  unsigned char    format;             /* reserved                          */
  AP_UINT16        primary_rc;         /* primary return code               */
  AP_UINT32        secondary_rc;       /* secondary return code             */
  unsigned char    data_lost;          /* previous indication lost          */
  unsigned char    ms_category[8];     /* Focal point category              */
  unsigned char    fp_fqcp_name[17];   /* Fully qualified focal point cp name*/
  unsigned char    ms_appl_name[8];    /* Focal point application name      */
  unsigned char    fp_type;            /* type of current focal point       */
  unsigned char    fp_status;          /* status of focal point             */
  unsigned char    fp_routing;         /* type of MDS routing to reach FP   */
  unsigned char    reserva[20];        /* reserved                          */
} FOCAL_POINT_INDICATION;
```

Parameters

opcode AP_FOCAL_POINT_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous focal point indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous focal point indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous focal point indications were lost.

ms_category Management Services category for which the focal point has changed. This can be either one of the category names specified in the MS Discipline-Specific Application Programs table contained in the *SNA Management Services Reference*, padded on the right with spaces if the name is shorter than 8 bytes, or a user-defined category. User-defined category names should be an 8-byte type-1134 EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes.

fp_fqcp_name Fully qualified name of the current focal point for the specified MS category. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters. If this parameter is set to 17 binary zeros, this indicates that there is no focal point currently defined for the specified MS category; the previous focal point has been deleted and not replaced.

ms_appl_name Name of the current focal point application. This can be either one of the application names specified in the MS Discipline-Specific Application Programs table in *Systems Network Architecture: Management Services Reference SC30-3346*, padded on the right with spaces to 8 bytes, or a user-defined application name (see "Related Publications"). User-defined names should be an 8-byte type-1134 EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes. If this parameter is set to 8 binary zeros, this indicates that there is no focal point currently defined for the specified MS category; the previous focal point has been deleted and not replaced.

fp_type Type of focal point. Refer to SNA Management Services for further detail. Possible values are:

AP_EXPLICIT_PRIMARY_FP

AP_IMPLICIT_PRIMARY_FP

AP_BACKUP_FP

AP_DEFAULT_PRIMARY_FP

AP_DOMAIN_FP

AP_HOST_FP

AP_NO_FP

fp_status Status of the focal point. Possible values are:

AP_NOT_ACTIVE

The focal point has gone from active to inactive.

AP_ACTIVE

The focal point has gone from inactive or pending active to active.

fp_routing Type of routing that applications should specify when sending data to the focal point. This parameter is used only if the focal point status is AP_ACTIVE. Possible values are:

AP_DEFAULT

Data should be sent using default routing.

AP_DIRECT

Data should be sent using direct routing.

LOCAL_LU_INDICATION

This indication is generated when a local LU is defined or deleted. This indication can be used by a registered application to maintain a list of all local LUs currently defined.

VCB Structure

```
typedef struct local_lu_indication
{
   AP_UINT16        opcode;             /* Indication operation code        */
   unsigned char    reserv2;            /* reserved                         */
   unsigned char    format;             /* reserved                         */
   AP_UINT16        primary_rc;         /* primary return code              */
   AP_UINT32        secondary_rc;       /* secondary return code            */
   unsigned char    data_lost;          /* Previous indication lost?        */
   unsigned char    reason;             /* reason for this indication       */
   unsigned char    lu_name[8];         /* LU name                          */
   DESCRIPTION      description;        /* resource description             */
   unsigned char    lu_alias[8];        /* LU alias                         */
   unsigned char    nau_address;        /* NAU address                      */
   unsigned char    reserv4;            /* reserved                         */
   unsigned char    pu_name[8];         /* PU name                          */
   unsigned char    lu_sscp_active;     /* Is LU-SSCP session active        */
   unsigned char    reserv5;            /* reserved                         */
   SESSION_STATS    lu_sscp_stats;      /* LU-SSCP session statistics       */
   unsigned char    sscp_id[6];         /* SSCP ID                          */
 } LOCAL_LU_INDICATION;

typedef struct session_stats
{
   AP_UINT16        rcv_ru_size;        /* session receive RU size          */
   AP_UINT16        send_ru_size;       /* session send RU size             */
   AP_UINT16        max_send_btu_size;  /* max send BTU size                */
   AP_UINT16        max_rcv_btu_size;   /* max receive BTU size             */
   AP_UINT16        max_send_pac_win;   /* max send pacing window size      */
   AP_UINT16        cur_send_pac_win;   /* current send pacing window size  */
   AP_UINT16        max_rcv_pac_win;    /* max receive pacing window size   */
   AP_UINT16        cur_rcv_pac_win;    /* current rcv pacing window size   */
   NB_COUNTER       send_data_frames;   /* number of data frames sent       */
   NB_COUNTER       send_fmd_data_frames; /* num of fmd data frames sent    */
   NB_COUNTER       send_data_bytes;    /* number of data bytes sent        */
   NB_COUNTER       rcv_data_frames;    /* number of data frames received   */
```

```
   NB_COUNTER       rcv_fmd_data_frames; /* num of fmd data frames received  */
   NB_COUNTER       rcv_data_bytes;      /* number of data bytes received    */
   unsigned char    sidh;                /* session ID high byte             */
   unsigned char    sidl;                /* session ID low byte              */
   unsigned char    odai;                /*origin-destination assignor bit set*/
   unsigned char    ls_name[8];          /* link station name                */
   unsigned char    pacing_type          /* type of pacing in use            */
} SESSION_STATS;
```

The LU-SSCP statistics contained in the session_stats structure are valid only when both the *nau_address* parameter is set to a nonzero value and the *lu_sscp_active* parameter is set to AP_YES. Otherwise, the parameters in the session_stats structure are reserved.

Parameters

opcode

AP_LOCAL_LU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous directory indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous directory indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous directory indications were lost.

reason

Reason for the indication. Possible values are:

AP_ADDED

The LU has been defined.

AP_REMOVED

The LU has been deleted, either explicitly using DELETE_LOCAL_LU, or implicitly, using DELETE_LS, DELETE_PORT, or DELETE_DLC.

AP_SSCP_ACTIVE

The LU-SSCP session has become active after the node has successfully processed an ACTLU.

AP_SSCP_INACTIVE

The LU-SSCP session has become inactive after a normal DACTLU or a link failure.

lu_name

Name of the local logical unit (LU) whose state has changed. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces.

description

Resource description, as specified on DEFINE_LOCAL_LU.

lu_alias

Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All eight bytes are significant.

nau_address

Network accessible unit (NAU) address of the LU. This value must be in the range 1-255. A nonzero value implies that the LU is a dependent LU. The value 0 (zero) implies that the LU is an independent LU.

pu_name

Name of the physical unit (PU) that this LU uses. This is an 8-byte type-A EBCDIC string), padded on the right with EBCDIC spaces. This parameter is significant only if the *nau_address* parameter is not set to 0 (zero). If the *nau_address* parameter is set to 0, the *pu_name* parameter is set to all binary zeros.

lu_sscp_sess_active

Specifies whether the LU-SSCP session is active. If the *nau_address* parameter is set to 0 (zero), this parameter is reserved. Possible values are:

AP_YES

The LU-SSCP session is active.

AP_NO

The LU-SSCP session is not active.

lu_sscp_stats.rcv_ru_size

This parameter is always reserved.

lu_sscp_stats.send_ru_size

This parameter is always reserved.

lu_sscp_stats.max_send_btu_size

Maximum basic transmission unit (BTU) that can be sent.

lu_sscp_stats.max_rcv_btu_size

Maximum basic transmission unit (BTU) that can be received.

lu_sscp_stats.max_send_pac_win

This parameter is always set to zero.

lu_sscp_stats.cur_send_pac_win

This parameter is always set to zero.

lu_sscp_stats.max_rcv_pac_win

This parameter is always set to zero.

lu_sscp_stats.cur_rcv_pac_win

This parameter is always set to zero.

lu_sscp_stats.send_data_frames

Number of normal flow data frames sent.

lu_sscp_stats.send_fmd_data_frames

Number of normal flow function management data

(FMD) frames sent.

lu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

lu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

lu_sscp_stats.rcv_fmd_data_frames

Number of normal flow function management data (FMD) frames received.

lu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

lu_sscp_stats.sidh

Session ID high byte.

lu_sscp_stats.sidl

Session ID low byte.

lu_sscp_stats.odai

Origin Destination Assignor indicator. When activating a session, the sender of the ACTLU sets this parameter to zero if the local node contains the primary link station and sets it to one if the ACTLU sender is the node containing the secondary link station.

lu_sscp_stats.ls_name

Link station name associated with statistics. This is an 8-byte string in a locally displayable character set. All eight bytes are significant. The parameter can be used to correlate this session with the link over which the session flows.

lu_sscp_stats.pacing_type

Receive pacing type in use on the LU-SSCP session. This will take the value AP_NONE.

sscp_id

The identifier of the SSCP as received in the ACTPU for the PU used by this LU. This parameter is 6 bytes

and is used only by dependent LUs. This parameter is set to all zeros for independent LUs or if the *lu_sscp_sess_active* parameter is not set to AP_YES.

LOCAL_TOPOLOGY_INDICATION

This indication is generated when a transmission group (TG) entry in a node's local topology database is activated or deactivated.

VCB Structure

```
typedef struct local_topol
ogy_indication
{
   AP_UINT16       opcode;             /* verb operation code              */
   unsigned char   reserv2;            /* reserved                         */
   unsigned char   format;             /* reserved                         */
   AP_UINT16       primary_rc;         /* primary return code              */
   AP_UINT32       secondary_rc;       /* secondary return code            */
   unsigned char   data_lost;          /* previous indication lost         */
   unsigned char   status;             /* TG status                        */
   unsigned char   dest[17];           /* name of TG destination node      */
   unsigned char   dest_type;          /* TG destination node type         */
   unsigned char   tg_num;             /* TG number                        */
   unsigned char   reserva[20];        /* reserved                         */
} LOCAL_TOPOLOGY_INDICATION;
```

Parameters

opcode AP_LOCAL_TOPOLOGY_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous local topology indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous local topology indications were lost.

AP_NO

	No previous local topology indications were lost.
status	Specifies the status of the TG. This can be AP_NONE or one or more of the following values (combined using a logical OR):
	AP_TG_OPERATIVE
	AP_TG_CP_CP_SESSIONS
	AP_TG_QUIESCING
dest	Fully qualified destination node name for the TG. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.
dest_type	Type of the destination node. Possible values are:
	AP_END_NODE
	End node.
	AP_NETWORK_NODE
	Network node.
	AP_VRN
	Virtual routing node.
tg_num	Transmission group number associated with the TG.

LS_INDICATION

This indication is generated when a link station is activated or deactivated. When the link station is deactivated, the returned data includes statistics on the link station's usage.

VCB Structure

```
typedef struct ls_indication
{
  AP_UINT16       opcode;              /* verb operation code               */
  unsigned char   reserv2;             /* reserved                          */
  unsigned char   format;              /* reserved                          */
  AP_UINT16       primary_rc;          /* primary return code               */
  AP_UINT32       secondary_rc;        /* secondary return code             */
  unsigned char   data_lost;           /* previous indication lost          */
  unsigned char   deactivated;         /* has LS been deactivated?          */
  unsigned char   ls_name[8];          /* link station name                 */
  unsigned char   description[32];     /* resource description              */
  unsigned char   reserv1[16];         /* reserved                          */
  unsigned char   adj_cp_name[17];     /* network qualified Adjacent CP name*/
  unsigned char   adj_node_type;       /* adjacent node type                */
  AP_UINT16       act_sess_count;      /* active session count on link      */
  unsigned char   indication_cause;    /* cause of indication               */
  LS_STATS        ls_stats;            /* link station statistics           */
  unsigned char   tg_num;              /* tg number                         */
  AP_UINT32       sense_data;          /* sense data                        */
  unsigned char   reserva[19];         /* reserved                          */
} LS_INDICATION;

typedef struct ls_stats
{
  AP_UINT32       in_xid_bytes;          /* number of XID bytes received     */
  AP_UINT32       in_msg_bytes;          /* number of message bytes received */
  AP_UINT32       in_xid_frames;         /* number of XID frames received    */
  AP_UINT32       in_msg_frames;         /* number of message frames received*/
  AP_UINT32       out_xid_bytes;         /* number of XID bytes sent         */
  AP_UINT32       out_msg_bytes;         /* number of message bytes sent     */
  AP_UINT32       out_xid_frames;        /* number of XID frames sent        */
  AP_UINT32       out_msg_frames;        /* number of message frames sent    */
  AP_UINT32       in_invalid_sna_frames;        /* number of invalid         */
                                                /*  frames received          */
  AP_UINT32       in_session_control_frames;    /* number of control         */
```

```
                                        /*  frames received          */
  AP_UINT32       out_session_control_frames;   /* number of control         */
                                        /*  frames sent              */
  AP_UINT32       echo_rsps;            /* reserved                           */
  AP_UINT32       current_delay;        /* reserved                           */
  AP_UINT32       max_delay;            /* reserved                           */
  AP_UINT32       min_delay;            /* reserved                           */
  AP_UINT32       max_delay_time;       /* reserved                           */
  AP_UINT32       good_xids;            /* successful XID on LS count         */
  AP_UINT32       bad_xids;             /* unsuccessful XID on LS count       */
} LS_STATS;
```

Parameters

opcode

AP_LS_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous LS indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous LS indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous LS indications were lost.

deactivated

Specifies whether the LS has been deactivated or activated. Possible values are:

AP_YES

The LS has been deactivated.

AP_NO

The LS has been activated.

ls_name

Name of the link station. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

description

A null-terminated text string describing the LS, as specified in the definition of the LS.

adj_cp_name

Fully qualified CP name of the adjacent node. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

adj_node_type

Type of the adjacent node. Possible values are:

AP_END_NODE

End node.

AP_NETWORK_NODE

Network node.

AP_LEN_NODE

LEN node.

AP_VRN

Virtual routing node.

act_sess_count

Total number of active sessions (both endpoint and intermediate) using the link.

indication_cause

Cause of the indication. Possible values are:

AP_ACTIVATING

The LS has been activated.

AP_DEACTIVATING

The LS has been deactivated.

AP_SESS_COUNT_CHANGING

The number of active sessions using the LS has changed.

AP_CP_NAME_CHANGING

The adjacent node's CP name has changed.

AP_DATA_LOST

A previous indication could not be sent.

AP_FAILED

The LS has failed.

AP_ACTIVATION_STARTED

The LS supports auto-activation, and has been started automatically when required for a session.

AP_ACTIVATION_FAILED

The LS supports auto-activation, but the attempt to start it automatically when required has failed.

AP_LR_ACTIVATING

The LS has failed (or an attempt to activate it has failed), and SNAplus2 is attempting to reactivate it.

The following parameters are returned only if deactivated is set to AP_YES, indicating that the LS has been deactivated.

ls_stats.in_xid_bytes

Total number of XID (Exchange Identification) bytes received on this link station.

ls_stats.in_msg_bytes

Total number of data bytes received on this link station.

ls_stats.in_xid_frames

Total number of XID (Exchange Identification) frames received on this link station.

ls_stats.in_msg_frames

Total number of data frames received on this link station.

ls_stats.out_xid_bytes

Total number of XID (Exchange Identification) bytes sent on this link station.

ls_stats.out_msg_bytes

Total number of data bytes sent on this link station.

ls_stats.out_xid_frames

Total number of XID (Exchange Identification) frames sent on this link station.

ls_stats.out_msg_frames

Total number of data frames sent on this link station.

ls_stats.in_invalid_sna_frames

Total number of SNA frames that were not valid received on this link station.

ls_stats.in_session_control_frames

Total number of session control frames received on this link station.

ls_stats.out_session_control_frames

Total number of session control frames sent on this link station.

ls_stats.good_xids

Total number of successful XID exchanges that have occurred on this link station since it was started.

ls_stats.bad_xids

Total number of unsuccessful XID exchanges that have occurred on this link station since it was started.

tg_num

Transmission group number associated with the LS.

sense_data

If the LS has failed because of an XID protocol error, this parameter contains the sense data associated with the error. If *indication_cause* is set to any value other than AP_FAILED, this parameter is reserved.

LU_0_TO_3_INDICATION

This indication is generated when the session status of a type 0-3 LU changes.

VCB Structure

```
typedef struct lu_0_to_3_indication
{
  AP_UINT16         opcode;                  /* verb operation code         */
  unsigned char     reserv2;                 /* reserved                    */
  unsigned char     format;                  /* reserved                    */
  AP_UINT16         primary_rc;              /* primary return code         */
  AP_UINT32         secondary_rc;            /* secondary return code       */
  unsigned char     data_lost;               /* previous indication lost    */
  unsigned char     pu_name[8];              /* PU Name                     */
  unsigned char     lu_name[8];              /* LU Name                     */
  unsigned char     description[32];          /* resource description       */
  unsigned char     reserv1[16];             /* reserved                    */
  unsigned char     nau_address;             /* NAU address                 */
  unsigned char     lu_sscp_sess_active;     /* Is SSCP session active?     */
  unsigned char     appl_conn_active;        /* Is application using LU?    */
  unsigned char     plu_sess_active;         /* Is PLU-SLU session active?  */
  unsigned char     host_attachment;         /* Host attachment             */
  SESSION_STATS     lu_sscp_stats;           /* LU-SSCP session statistics  */
  SESSION_STATS     plu_stats;               /* PLU session statistics      */
} LU_0_TO_3_INDICATION;


typedef struct session_stats
{
  AP_UINT16        rcv_ru_size;          /* session receive RU size          */
  AP_UINT16        send_ru_size;         /* session send RU size             */
  AP_UINT16        max_send_btu_size;    /* maximum send BTU size            */
  AP_UINT16        max_rcv_btu_size;     /* maximum rcv BTU size             */
  AP_UINT16        max_send_pac_win;     /* maximum send pacing window size  */
  AP_UINT16        cur_send_pac_win;     /* current send pacing window size  */
  AP_UINT16        max_rcv_pac_win;      /* maximum receive pacing window    */
                                         /*  size                            */
  AP_UINT16        cur_rcv_pac_win;      /* current receive pacing window    */
                                         /*  size                            */
  AP_UINT32        send_data_frames;     /* number of data frames sent       */
  AP_UINT32        send_fmd_data_frames; /* num fmd data frames sent         */
  AP_UINT32        send_data_bytes;      /* number of data bytes sent        */
```

```
  AP_UINT32        rcv_data_frames;      /* number of data frames received   */
  AP_UINT32        rcv_fmd_data_frames;  /* num fmd data frames received     */
  AP_UINT32        rcv_data_bytes;       /* number of data bytes received    */
  unsigned char    sidh;                 /* session ID high byte (from LFSID)*/
  unsigned char    sidl;                 /* session ID low byte (from LFSID) */
  unsigned char    odai;                 /* ODAI bit set                     */
  unsigned char    ls_name[8];           /* Link station name                */
  unsigned char    reserve;              /* reserved                         */
} SESSION_STATS;
```

Parameters

opcode

AP_LU_0_TO_3_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous LU 0-3 indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous LU 0-3 indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous LU 0-3 indications were lost.

pu_name

Name of the local PU used by the LU. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the name is shorter than 8 characters.

lu_name

Name of the LU whose session status has changed.

This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

description

A null-terminated text string describing the LU, as specified in the definition of the LU.

nau_address

Network accessible unit address of the LU.

lu_sscp_sess_active

Specifies whether the SSCP session is active-that is, whether the ACTLU has been successfully processed. Possible values are:

AP_YES

The session is active.

AP_NO

The session is not active.

appl_conn_active

Specifies whether an application is using the LU. Possible values are:

AP_YES

An application is using the LU.

AP_NO

No application is using the LU.

plu_sess_active

Specifies whether the PLU-SLU session has been activated. Possible values are:

AP_YES

The session is active.

AP_NO

The session is not active.

host_attachment

LU host attachment type.

Possible values are:

AP_DIRECT_ATTACHED

LU is directly attached to the host system.

AP_DLUR_ATTACHED

LU is attached to the host system using DLUR

A session_stats structure is included for each of the two sessions (LU-SSCP session and PLU-SLU session). If the session goes from active to inactive, the structure contains the following parameters; otherwise these parameters are reserved.

rcv_ru_size

Maximum receive RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

send_ru_size

Maximum send RU size. (In the LU-SSCP session statistics, this parameter is reserved.)

max_send_btu_size

Maximum BTU size that can be sent.

max_rcv_btu_size

Maximum BTU size that can be received.

max_send_pac_win

Maximum size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_send_pac_win

Current size of the send pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

max_rcv_pac_win

Maximum size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

cur_rcv_pac_win

Current size of the receive pacing window on this session. (In the LU-SSCP session statistics, this parameter is reserved.)

send_data_frames

Number of normal flow data frames sent.

send_fmd_data_frames

Number of normal flow FMD data frames sent.

send_data_bytes

Number of normal flow data bytes sent.

rcv_data_frames

Number of normal flow data frames received.

rcv_fmd_data_frames

Number of normal flow FMD data frames received.

rcv_data_bytes

Number of normal flow data bytes received.

sidh

Session ID high byte.

sidl

Session ID low byte.

odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station.

ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters.

MODE_INDICATION

This indication is sent when a local LU and partner LU start to communicate using a particular mode, or when the active session count for the LU-LU-mode combination changes.

VCB Structure

```
typedef struct mode_indication
{
  AP_UINT16        opcode;              /* verb operation code              */
  unsigned char    reserv2;             /* reserved                         */
  unsigned char    format;              /* reserved                         */
  AP_UINT16        primary_rc;          /* primary return code              */
  AP_UINT32        secondary_rc;        /* secondary return code            */
  unsigned char    data_lost;           /* previous indication lost         */
  unsigned char    removed;             /* is entry being removed?          */
  unsigned char    lu_alias[8];         /* LU alias                         */
  unsigned char    plu_alias[8];        /* partner LU alias                 */
  unsigned char    fqplu_name[17];      /* fully qualified partner LU name  */
  unsigned char    mode_name[8];        /* mode name                        */
  unsigned char    description[32];     /* resource description             */
  unsigned char    reserv1[16];         /* reserved                         */
  AP_UINT16        curr_sess_count;     /* current session count            */
  unsigned char    reserva[20];         /* reserved                         */
} MODE_INDICATION;
```

Parameters

opcode

AP_MODE_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous mode indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting

the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous mode indications were lost.

AP_NO

No previous mode indications were lost.

removed

This parameter is currently not used; a mode indication is generated only when the LUs start to use the mode, and not when they stop using it.

lu_alias

Locally defined LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

fqplu_name

Fully qualified name of the partner LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

mode_name

Mode name which designates the network properties for a group of sessions. This is an 8-byte type-A EBCDIC string (starting with a letter), padded on the right with spaces if the name is shorter than 8 characters.

description

A null-terminated text string describing the mode, as specified in the definition of the mode.

curr_sess_count

The number of sessions currently active for this LU-LU-mode combination.

NOF_STATUS_INDICATION

This indication is generated when the application can no longer access its connected target (because the SNAplus2 software on the target computer has been stopped, or because the communications path to the target computer has failed). If the target is the domain configuration file, it is also generated if another server takes over as master (and therefore the connected target file is no longer the master copy of the file).

The application does not need to register explicitly for this indication. SNAplus2 will return it to any application that has registered for any type of indications on the specified target handle. If the application is currently registered to receive indications using more than one callback routine, SNAplus2 returns this indication to the first routine registered.

After the application receives an indication that the target can no longer be accessed, all subsequent verbs using the relevant target handle will be rejected, apart from DISCONNECT_NODE or CLOSE_FILE (to end the application's connection to the target). In addition, any registrations for indications on this target handle will be lost; in order to continue receiving indications when the target becomes available, the application must reconnect to the target and reregister for the required indications.

VCB Structure

```
typedef struct nof_status_indication
{
   AP_UINT16        opcode;                /* verb operation code          */
   unsigned char    reserv2;               /* reserved                     */
   unsigned char    format;                /* reserved                     */
   AP_UINT16        primary_rc;            /* primary return code          */
   AP_UINT32        secondary_rc;          /* secondary return code        */
   AP_UINT32        status;                /* status being reported        */
   AP_UINT32        dead_target_handle;    /* Handle of dead connection    */
                                           /*  NULL for system termination */
   unsigned char    reserv1[32];           /* reserved                     */
} NOF_STATUS_INDICATION;
```

Parameters

opcode

AP_NOF_STATUS_INDICATION

primary_rc

AP_OK

status

Specifies the status change being reported. Possible values are:

AP_LOCAL_ABENDED

The SNAplus2 software on the local computer has stopped. The application should not attempt to issue any more NOF verbs until the software has been restarted.

AP_TARGET_ABENDED

The SNAplus2 software on the target computer has stopped or the communications path to it has failed.

AP_MASTER_TAKEOVER

This value is returned only when the application is connected to the master configuration file (specified by the *requested_role* parameter on OPEN_FILE). Another server has now taken over as master, so the target file is no longer the master configuration file. If the application needs to make further changes to the running configuration, it must use CLOSE_FILE to end its connection with the file, and then issue OPEN_FILE again to access the new master configuration file.

dead_target_handle

The target handle of the failed target or of the file that is no longer the master configuration file. The application should not attempt to issue any further verbs for this target handle except DISCONNECT_NODE or CLOSE_FILE.

If *status* is set to AP_LOCAL_ABENDED, this parameter is reserved.

PLU_INDICATION

This indication is generated when a local LU begins to communicate with a partner LU. This occurs either when the first ALLOCATE to this PLU is processed or when the first BIND is received from this PLU. The indication is also generated if the partner LU's CP name changes.

VCB Structure

```
typedef struct plu_indication
{
   AP_UINT16        opcode;                 /* verb operation code        */
   unsigned char    reserv2;                /* reserved                   */
   unsigned char    format;                 /* reserved                   */
   AP_UINT16        primary_rc;             /* primary return code        */
   AP_UINT32        secondary_rc;           /* secondary return code      */
   unsigned char    data_lost;              /* has previous indication    */
                                            /*  been lost?                */
   unsigned char    removed;                /* is entry being removed?    */
   unsigned char    lu_alias[8];            /* LU alias                   */
   unsigned char    plu_alias[8];           /* partner LU alias           */
   unsigned char    fqplu_name[17];         /* fully qualified partner    */
                                            /*  LU name                   */
   unsigned char    description[32];        /* resource description       */
   unsigned char    reserv1[16];            /* reserved                   */
   unsigned char    partner_cp_name[17];    /* partner CP name            */
   unsigned char    partner_lu_located;     /* partner CP name resolved?  */
   unsigned char    reserva[20];            /* reserved                   */
} PLU_INDICATION;
```

Parameters

opcode

AP_PLU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous PLU indications have

been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous PLU indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous PLU indications were lost.

removed

This parameter is currently not used; a PLU indication is generated only when the LUs start to communicate, and not when they stop communicating.

lu_alias

Local LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

plu_alias

Partner LU alias. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

fqplu_name

17-byte fully qualified network name for the partner LU. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

description

A null-terminated text string describing the partner LU, as specified in the definition of the partner LU.

partner_cp_name

17-byte fully qualified network name for the CP

associated with the partner LU. This parameter is not used if *partner_lu_located* below is set to AP_NO.

The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

partner_lu_located

Specifies whether the local node has located the CP where the partner LU is located. Possible values are:

AP_YES

The partner LU has been located. The *partner_cp_name* parameter contains the CP name of the partner LU.

AP_NO

The partner LU has not yet been located. The *partner_cp_name* parameter should not be checked.

PORT_INDICATION

This indication is generated when a port is activated or deactivated.

VCB Structure

```
typedef struct port_indication
{
   AP_UINT16        opcode;              /* verb operation code           */
   unsigned char    reserv2;             /* reserved                      */
   unsigned char    format;              /* reserved                      */
   AP_UINT16        primary_rc;          /* primary return code           */
   AP_UINT32        secondary_rc;        /* secondary return code         */
   unsigned char    data_lost;           /* previous indication lost      */
   unsigned char    deactivated;         /* has session been deactivated? */
   unsigned char    port_name[8];        /* port name                     */
   unsigned char    description[32];     /* resource description          */
   unsigned char    reserv1[16];         /* reserved                      */
   unsigned char    reserva[20];         /* reserved                      */
} PORT_INDICATION;
```

Parameters

opcode

AP_PORT_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous port indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous port indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous port indications were lost.

deactivated

Specifies whether the port has been deactivated or activated. Possible values are:

AP_YES

The port has been deactivated.

AP_NO

The port has been activated.

port_name

Name of port. This is an 8-byte ASCII string, padded on the right with spaces if the name is shorter than 8 bytes.

description

A null-terminated text string describing the port, as specified in the definition of the port.

PU_INDICATION

This indication is generated when the PU-SSCP session status of a local PU changes.

VCB Structure

```
typedef struct pu_indication
{
   AP_UINT16         opcode;                 /* verb operation code        */
   unsigned char     reserv2;                /* reserved                   */
   unsigned char     format;                 /* reserved                   */
   AP_UINT16         primary_rc;             /* primary return code        */
   AP_UINT32         secondary_rc;           /* secondary return code      */
   unsigned char     data_lost;              /* previous indication lost   */
   unsigned char     pu_name[8];             /* PU Name                    */
   unsigned char     description[32];        /* resource description       */
   unsigned char     reserv3[16];            /* reserved                   */
   unsigned char     pu_sscp_sess_active;    /* Is SSCP session active?    */
   unsigned char     host_attachment;        /* Host attachment            */
   unsigned char     reserv1[2];             /* reserved                   */
   SESSION_STATS     pu_sscp_stats;          /* PU-SSCP session statistics */
} PU_INDICATION;


typedef struct session_stats
{
  AP_UINT16        rcv_ru_size;           /* session receive RU size         */
  AP_UINT16        send_ru_size;          /* session send RU size            */
  AP_UINT16        max_send_btu_size;     /* maximum send BTU size           */
  AP_UINT16        max_rcv_btu_size;      /* maximum rcv BTU size            */
  AP_UINT16        max_send_pac_win;      /* maximum send pacing window size */
  AP_UINT16        cur_send_pac_win;      /* current send pacing window size */
  AP_UINT16        max_rcv_pac_win;       /* maximum receive pacing          */
                                          /*  window size                    */
  AP_UINT16        cur_rcv_pac_win;       /* current receive pacing          */
                                          /*  window size                    */
  AP_UINT32        send_data_frames;      /* number of data frames sent      */
  AP_UINT32        send_fmd_data_frames;  /* num fmd data frames sent        */
  AP_UINT32        send_data_bytes;       /* number of data bytes sent       */
  AP_UINT32        rcv_data_frames;       /* number of data frames received  */
  AP_UINT32        rcv_fmd_data_frames;   /* num fmd data frames received    */
  AP_UINT32        rcv_data_bytes;        /* number of data bytes received   */
  unsigned char    sidh;                  /* session ID high byte            */
```

```
                                        /*  (from LFSID)                    */
  unsigned char    sidl;                /* session ID low byte (from LFSID)*/
  unsigned char    odai;                /* ODAI bit set                    */
  unsigned char    ls_name[8];          /* Link station name               */
  unsigned char    reserve;             /* reserved                        */
} SESSION_STATS;
```

Parameters

opcode

AP_PU_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous PU indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous PU indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous PU indications were lost.

pu_name

Name of the PU (specified on the DEFINE_LS verb). This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 characters.

description

A null-terminated text string describing the PU, as specified in the definition of the PU.

pu_sscp_sess_active

Specifies whether the PU-SSCP session is active (whether the ACTPU has been successfully processed). Possible values are:

AP_YES

The PU-SSCP session is active.

AP_NO

The PU-SSCP session is inactive.

host_attachment

Local PU host attachment type.

Possible values are:

AP_DIRECT_ATTACHED

PU is directly attached to the host system.

AP_DLUR_ATTACHED

PU is supported by DLUR.

The following parameters are used only when the session state changes from active to inactive:

pu_sscp_stats.rcv_ru_size

Reserved (always set to zero).

pu_sscp_stats.send_ru_size

Reserved (always set to zero).

pu_sscp_stats.max_send_btu_size

Maximum BTU size that can be sent.

pu_sscp_stats.max_rcv_btu_size

Maximum BTU size that can be received.

pu_sscp_stats.max_send_pac_win

Reserved (always set to zero).

pu_sscp_stats.cur_send_pac_win

Reserved (always set to zero).

pu_sscp_stats.max_rcv_pac_win

Reserved (always set to zero).

pu_sscp_stats.cur_rcv_pac_win

Reserved (always set to zero).

pu_sscp_stats.send_data_frames

Number of normal flow data frames sent.

pu_sscp_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

pu_sscp_stats.send_data_bytes

Number of normal flow data bytes sent.

pu_sscp_stats.rcv_data_frames

Number of normal flow data frames received.

pu_sscp_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

pu_sscp_stats.rcv_data_bytes

Number of normal flow data bytes received.

pu_sscp_stats.sidh

Session ID high byte.

pu_sscp_stats.sidl

Session ID low byte.

pu_sscp_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station and sets it to one if the BIND sender is the node containing the secondary link station.

pu_sscp_stats.ls_name

Link station name associated with statistics. This is an 8-byte ASCII character string, right-padded with spaces if the name is shorter than 8 characters.

REGISTRATION_FAILURE

REGISTRATION_FAILURE indicates that an attempt to register resources with the network node server failed.

VCB Structure

```
typedef struct registration_failure
{
   AP_UINT16        opcode;              /* verb operation code             */
   unsigned char    reserv2;             /* reserved                        */
   unsigned char    format;              /* reserved                        */
   AP_UINT16        primary_rc;          /* primary return code             */
   AP_UINT32        secondary_rc;        /* secondary return code           */
   unsigned char    data_lost;           /* previous indication lost        */
   unsigned char    resource_name[17];   /* network qualified resource name */
   AP_UINT16        resource_type;       /* resource type                   */
   unsigned char    description[32];     /* resource description            */
   unsigned char    reserv1[16];         /* reserved                        */
   unsigned char    reserv2b[2];         /* reserved                        */
   AP_UINT32        sense_data;          /* sense data                      */
   unsigned char    reserva[20];         /* reserved                        */
} REGISTRATION_FAILURE;
```

Parameters

opcode AP_REGISTRATION_FAILURE

primary_rc AP_OK

data_lost Specifies whether any previous registration failure indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous registration failure indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous registration failure indications were lost.

resource_name Name of resource that failed to register. The name is a 17-byte EBCDIC string, right-padded with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

resource_type Resource type of resource that failed to register. One of the following.

AP_NNCP_RESOURCE

Network node.

AP_ENCP_RESOURCE

End node.

AP_LU_RESOURCE

LU.

description A null-terminated text string describing the resource, as specified in the definition of the resource.

sense_data Sense data (specified in SNA Formats).

SERVER_INDICATION

This indication is generated when the SNAplus2 software is started or stopped on another computer on the LAN or when a server's role as master or backup server changes. A NOF application can use these indications to keep track of which servers are currently active or to determine when a new server has successfully taken over as master.

Server indications are also generated (for SNAplus2 internal use) when the status of other SNAplus2 components on a server changes. If the application needs to use server indications as described above, it should check the *status* and *flags* parameters for changes; it can ignore any server indications where these parameters do not indicate a change.

The REGISTER_INDICATION_SINK verb used to register for server indications should be issued with a null target handle; it is not associated with any particular target.

VCB Structure

```
typedef struct server_indication
{
    AP_UINT16        opcode;                /* verb operation code          */
    unsigned char    reserv2;               /* reserved                     */
    unsigned char    format;                /* reserved                     */
    AP_UINT16        primary_rc;            /* primary return code          */
    AP_UINT32        secondary_rc;          /* secondary return code        */
    unsigned char    data_lost;             /* previous indication lost     */
    AP_UINT32        status;                /* node status                  */
    AP_UINT32        flags;                 /* is server master or backup?  */
    unsigned char    server_name[64];       /* name of server               */
} SERVER_INDICATION;
```

Parameters

opcode AP_SERVER_INDICATION

primary_rc AP_OK

data_lost Specifies whether any previous server indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an

internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous server indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous server indications were lost.

status Specifies the status of the SNA software on the indicated server. Possible values are:

AP_ACTIVE

The SNA software has been started.

AP_NOT_ACTIVE

The SNA software has been stopped.

flags Specifies whether the indicated server is the master server or a backup server. The application should use a logical AND operation to check the appropriate values, as follows:

- If the expression "*flags* AND AP_MASTER_FLAG" is nonzero, the indicated server is the master server.
- If the expression "*flags* AND AP_BACKUP_FLAG" is nonzero, the indicated server is a backup server.

server_name Name of the server on which the SNA software has been started or stopped.

SESSION_INDICATION

This indication is generated when a session is activated or deactivated. When a session is deactivated, the verb returns statistics on the usage of the session.

VCB Structure

```
typedef struct session_indication
{
   AP_UINT16         opcode;             /* verb operation code             */
   unsigned char     reserv2;            /* reserved                        */
   unsigned char     format;             /* reserved                        */
   AP_UINT16         primary_rc;         /* primary return code             */
   AP_UINT32         secondary_rc;       /* secondary return code           */
   unsigned char     data_lost;          /* previous indication lost        */
   unsigned char     deactivated;        /* has session been deactivated?   */
   unsigned char     lu_name[8];         /* LU name                         */
   unsigned char     lu_alias[8];        /* LU alias                        */
   unsigned char     plu_alias[8];       /* partner LU alias                */
   unsigned char     fqplu_name[17];     /* fully qualified partner LU name */
   unsigned char     mode_name[8];       /* mode name                       */
   unsigned char     session_id[8];      /* session ID                      */
   FQPCID            fqpcid;             /* fully qualified procedure       */
                                         /*  correlator ID                  */
   AP_UINT32         sense_data;         /* sense data                      */
   unsigned char     duplex_support;     /* full-duplex support             */
   SESSION_STATS     sess_stats;         /* session statistics              */
   unsigned char     sscp_id[6];         /* SSCP ID                         */
   unsigned char     reserva[12];        /* reserved                        */
} SESSION_INDICATION;

typedef struct fqpcid
{
   unsigned char     pcid[8];            /* procedure correlator identifier */
   unsigned char     fqcp_name[17];      /* originator's network qualified  */
                                         /*  CP name                        */
   unsigned char     reserve3[3];        /* reserved                        */
} FQPCID;

typedef struct session_stats
{
```

```
  AP_UINT16        rcv_ru_size;           /* session receive RU size         */
  AP_UINT16        send_ru_size;          /* session send RU size            */
  AP_UINT16        max_send_btu_size;     /* maximum send BTU size           */
  AP_UINT16        max_rcv_btu_size;      /* maximum rcv BTU size            */
  AP_UINT16        max_send_pac_win;      /* maximum send pacing window size */
  AP_UINT16        cur_send_pac_win;      /* current send pacing window size */
  AP_UINT16        max_rcv_pac_win;       /* maximum receive pacing window   */
                                          /*  size                           */
  AP_UINT16        cur_rcv_pac_win;       /* current receive pacing window   */
                                          /*  size                           */
  AP_UINT32        send_data_frames;      /* number of data frames sent      */
  AP_UINT32        send_fmd_data_frames;  /* num fmd data frames sent        */
  AP_UINT32        send_data_bytes;       /* number of data bytes sent       */
  AP_UINT32        rcv_data_frames;       /* number of data frames received  */
  AP_UINT32        rcv_fmd_data_frames;   /* num fmd data frames received    */
  AP_UINT32        rcv_data_bytes;        /* number of data bytes received   */
  unsigned char    sidh;                  /* session ID high byte            */
                                          /*  (from LFSID)                   */
  unsigned char    sidl;                  /* session ID low byte (from LFSID)*/
  unsigned char    odai;                  /* ODAI bit set                    */
  unsigned char    ls_name[8];            /* Link station name               */
  unsigned char    pacing_type;           /* Pacing type                     */
} SESSION_STATS;
```

Parameters

opcode

AP_SESSION_INDICATION

primary_rc

AP_OK

data_lost

Specifies whether any previous session indications have been lost. If SNAplus2 detects a condition that prevents it from sending an indication (for example an internal resource shortage), it indicates this by setting the *data_lost* parameter on the next indication after the condition has cleared. Possible values are:

AP_YES

One or more previous session indications were lost. Later fields in this VCB may be set to zeros.

AP_NO

No previous session indications were lost.

deactivated

Specifies whether the session has been deactivated or activated. Possible values are:

AP_YES

The session has been deactivated.

AP_NO

The session has been activated.

lu_name

LU name of the local LU, as defined to SNAplus2. This is an 8-byte type-A EBCDIC string, padded on the right with spaces if the name is shorter than 8 bytes.

lu_alias

LU alias of the local LU, as defined to SNAplus2. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes.

plu_alias

LU alias of the partner LU. This is an 8-byte ASCII string, using any locally displayable characters, padded on the right with spaces if the name is shorter than 8 bytes.

fqplu_name

Fully qualified LU name for the partner LU, as defined to SNAplus2. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1-8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1-8 A-string characters.

mode_name

Name of the mode used by the LUs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded on the right with EBCDIC spaces if the

name is shorter than 8 bytes.

session_id

8-byte identifier of the session.

fqpcid.pcid

Procedure Correlator ID. This is an 8-byte hexadecimal string.

fqpcid.fqcp_name

Fully qualified CP name. The name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of up to 8 A-string characters, an EBCDIC dot (period) character, and a network name of up to 8 A-string characters.

The following parameters are used only if *deactivated* is set to AP_YES:

sense_data

The sense data sent or received on the UNBIND message that ended the session.

duplex_support

The conversation duplex support as negotiated on the BIND. Possible values are:

AP_HALF_DUPLEX

Only half-duplex conversations are supported.

AP_FULL_DUPLEX

Both half-duplex and full-duplex conversations are supported. Expedited data is also supported.

AP_UNKNOWN

Duplex support is not known because the session has deactivated.

sess_stats.rcv_ru_size

Maximum receive RU size.

sess_stats.send_ru_size

Maximum send RU size.

sess_stats.max_send_btu_size

Maximum BTU size that can be sent.

sess_stats.max_rcv_btu_size

Maximum BTU size that can be received.

sess_stats.max_send_pac_win

Maximum size of the send pacing window on this session.

sess_stats.cur_send_pac_win

Current size of the send pacing window on this session.

sess_stats.max_rcv_pac_win

Maximum size of the receive pacing window on this session.

sess_stats.cur_rcv_pac_win

Current size of the receive pacing window on this session.

sess_stats.send_data_frames

Number of normal flow data frames sent.

sess_stats.send_fmd_data_frames

Number of normal flow FMD data frames sent.

sess_stats.send_data_bytes

Number of normal flow data bytes sent.

sess_stats.rcv_data_frames

Number of normal flow data frames received.

sess_stats.rcv_fmd_data_frames

Number of normal flow FMD data frames received.

sess_stats.rcv_data_bytes

Number of normal flow data bytes received.

sess_stats.sidh

Session ID high byte.

sess_stats.sidl

Session ID low byte.

sess_stats.odai

Origin Destination Assignor Indicator. When bringing up a session, the sender of the BIND sets this field to zero if the local node contains the primary link station, and sets it to one if the BIND sender is the node containing the secondary link station.

sess_stats.ls_name

Link station name associated with statistics. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant. This field can be used to correlate the session statistics with the link over which session traffic flows.

sess_stats.pacing_type

The type of receive pacing in use on this session.

sscp_id

For dependent LU sessions, the identifier of the SSCP as received in the ACTPU for the PU used by this LU. This parameter is 6 bytes and is used only by dependent LUs. This parameter is set to all zeros for independent LUs.

SNA_NET_INDICATION

This indication is generated when another NOF application or a SNAplus2 administration tool makes a change to the SNA network file `sna.net`. The target for this verb, identified by the *`target_handle`* parameter on the REGISTER_INDICATION_SINK verb that registers to receive this indication, must be the `sna.net` file.

VCB Structure

No specific VCB structure is associated with this indication. To register for SNA network indications, the application specifies the value `AP_SNA_NET_INDICATION` as the *`indication_opcode`* parameter on REGISTER_INDICATION_SINK. When a change is made to the SNA network file, SNAplus2 then reports this to the application's callback routine by sending a copy of the VCB from the NOF verb (ADD_BACKUP or DELETE_BACKUP) that made the change.

To enable the application to distinguish between SNA network indications and asynchronous responses to its own NOF verbs issued to the SNA network file, SNAplus2 changes the *`primary_rc`* parameter in the VCB for an indication. The value `AP_INDICATION` identifies a VCB associated with an SNA network file indication; the value `AP_OK`, or any other value, indicates an asynchronous response to one of the application's own NOF verbs.

A Common Return Codes

Overview

This appendix describes the primary and secondary return codes that are common to all NOF verbs.

Return codes that are specific to a particular verb, or a group of verbs, are described in the individual verb descriptions in Chapter 6, “NOF Indications.”

To translate the hexadecimal return codes (for example, `0x58100000`) that a program returns into the names provided in this guide (for example, `AP_INVALID_SYM_DEST_NAME`), refer to the header file `values_c.h` provided on the machine. Then look up the verb definition in this guide to see what the return code means.

Communications Subsystem Not Active

If the verb does not execute because a required component is not active, SNAplus2 returns the following parameters:

primary_rc	AP_COMM_SUBSYSTEM_ABENDED
secondary_rc	One of the following:
	AP_LOCAL_ABENDED
	The SNAplus2 software has stopped.
	AP_TARGET_ABENDED
	The target node has stopped or the communication path to it has failed.
primary_rc	AP_COMM_SUBSYSTEM_NOT_LOADED
	The SNAplus2 software is not active.
secondary_rc	Not used.
primary_rc	AP_NODE_NOT_STARTED
	The target node has not been started.
secondary_rc	Not used.
primary_rc	AP_NODE_STOPPING
	The target node is in the process of stopping (as a result of a TERM_NODE verb).
secondary_rc	Not used.

Indication

This return code does not signify an error.

If the application has registered using REGISTER_INDICATION_SINK to receive configuration indications or SNA network file indications, SNAplus2 sends an indication each time another NOF API application or a SNAplus2 component modifies the target file or the target node's configuration. The format of this indication is the same as the returned VCB for the NOF verb that modified the configuration. SNAplus2 sets this primary return code to indicate that the VCB being returned is a configuration indication or an SNA network file indication, rather than the response to a verb issued by the application; this enables the application to distinguish between its own verb returns and indications resulting from verbs issued by other applications.

primary_rc AP_INDICATION

secondary_rc Possible values are:

AP_EXTRA_DATA_LOST

SNAplus2 was unable to allocate sufficient storage to return the complete VCB for this indication; the returned information is incomplete. The application should issue the appropriate QUERY_* verb to obtain more information about the modified component.

(zero)

The complete VCB for this indication is being returned.

Invalid Function

If the verb does not execute because the node does not recognize it as a valid verb, SNAplus2 returns the following parameters:

primary_rc AP_INVALID_VERB

The *opcode* parameter was not set to the operation code of any NOF verb or the verb identified by this parameter cannot be used because this version of SNAplus2 does not support it.

secondary_rc Not used.

primary_rc AP_FUNCTION_NOT_SUPPORTED

The NOF verb identified by the specified *opcode* parameter cannot be used because the target node's configuration does not support it.

secondary_rc Not used.

Parameter Check

If the verb does not execute because of a parameter error, SNAplus2 returns the following parameters:

primary_rc AP_PARAMETER_CHECK

secondary_rc One of the following:

AP_INVALID_FORMAT

The reserved parameter format was not set to zero.

AP_INVALID_TARGET_HANDLE

The supplied target handle is not valid.

AP_INVALID_TARGET

The verb cannot be issued to the specified target. For example, QUERY_PARTNER_LU, which returns information about an LU's current usage, can be issued only to a running node; it is not valid when issued to a file.

AP_INVALID_TARGET_MODE

The verb cannot be issued in the current mode. For example, only QUERY_* verbs can be issued in read-only mode; DEFINE_*, DELETE_*, START_*, and STOP_* verbs are not valid in this mode.

AP_SYNC_NOT_ENABLED

The application issued this verb within a callback routine, using the synchronous NOF entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check

If the verb does not execute because of a state check, SNAplus2 returns the following parameters.

primary_rc AP_STATE_CHECK

secondary_rc One of the following:

AP_CANT_MODIFY_VISIBILITY

You have attempted to define a resource with a name that is reserved for use internally by SNAplus2. Please choose a different name.

AP_FILE_LOCK_FAILED

The application issued SET_PROCESSING_MODE to change to commit mode, but SNAplus2 failed to get a lock on the configuration file. This can be because another NOF API application or SNAplus2 component is already accessing the file.

AP_FILE_UNLOCK_FAILED

The application issued SET_PROCESSING_MODE to change from commit mode to one of the other modes, but SNAplus2 failed to release its lock on the configuration file. In order to free the file when this error occurs, SNAplus2 closes the application's handle to the file. The application must issue OPEN_FILE again, to obtain a new file handle, before attempting to issue any more verbs to this file.

AP_FILE_UNAVAILABLE

The connection to the target file has been lost.

AP_NODE_NOT_CONFIGURED

The system has just been installed, and a DEFINE_NODE verb has not yet been issued (or the Motif or ascii administration tools have not yet been used to configure the node). You must do this before issuing any other verbs.

AP_NOT_MASTER

The target file is a copy of the domain configuration file, or of the `sna.net` file, on a server that is not the master server. Verbs that modify these files must be issued to the master server's copy of the files.

`AP_SYNC_PENDING`

This verb was issued using the synchronous NOF API entry point, but another synchronous verb was in progress. Only one synchronous verb can be in progress at any time.

System Error

If the verb does not execute because of an operating system error, SNAplus2 returns the following parameters:

primary_rc AP_UNEXPECTED_SYSTEM_ERROR

An operating system call failed during processing of the verb.

secondary_rc The secondary return code in this case is the return code from the operating system call. For the meaning of this return code, check the returned value in the file `errno.h` on the computer where the error occurred. Typically, the return code will indicate a condition such as memory shortage.

If the verb was issued to change the target configuration (such as DEFINE_* or DELETE_*), or to perform an action (such as START_*), the application should issue the appropriate QUERY_* verb to determine whether the change or action succeeded. In particular, if this error occurs while processing a DEFINE_* or DELETE_* verb containing multiple data structures(such as DEFINE_EMULATOR_USER), the change can be incomplete (for example, some but not all of the user's sessions may have been defined).

System Error

Index