Communications Server for Windows, Version 6.1
and
Personal Communications for Windows, Version 5.7

System Management Programming

SC31-8480-06

Communications Server for Windows, Version 6.1
and
Personal Communications for Windows, Version 5.7

System Management Programming

SC31-8480-06

Note

Before using this information and the product it supports, read the information in Appendix B, "Notices", on page 629.

Seventh Edition (September 2003)

This edition applies to Version 6.1 of IBM Communications Server for Windows, Version 5.7 of Personal Communications for Windows (program number: 5639–I70), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Tables

About This Book

This book describes how to develop programs that use IBM® Communications Server for Windows® and IBM Personal Communications for Windows.

IBM Communications Server for Windows (referred to as *Communications Server*) is a communications services platform. This platform provides a wide range of services for workstations that communicate with host computers and with other workstations. Communications Server users can choose from among a variety of remote connectivity options.

IBM Personal Communications for Windows (referred to as *Personal Communications*) is a full-function emulator. In addition to host terminal emulation, it provides these useful features:

- File transfer
- Dynamic configuration
- An easy-to-use graphical interface
- APIs for SNA-based client applications
- An API allowing TCP/IP-based applications to communicate over an SNA-based network.

While in most instances, developing programs for Personal Communications and Communications Server is very similar in that they each support many of the same verbs, there are some differences. These differences are denoted through the use of icons. See “Icons” on page x for specific details. Throughout this book, *the Program* refers to both Personal Communications and Communications Server. When only the Personal Communications program or only the Communications Server program applies, then that specific program name is used.

In this book, *Windows* refers to Windows 95, Windows 98, Windows NT®, Windows Me, Windows 2000, and Windows XP. Throughout this book, *workstation* refers to all supported personal computers. When only one model or architecture of the personal computer is referred to, only that type is specified.

Who Should Read This Book

This book is intended for programmers and developers who plan to use Node Operator Facility (NOF) API messages to manage and query the operation of Personal Communications or Communications Server, or plan to use ASCII Configuration files or both.

This book is also intended for developers who are writing network management applications that use the underlying management services support provided by Personal Communications and Communications Server to communicate with remote (host focal point) network management applications.

How to Use This Book

This book is organized into two parts. Part 1, “Personal Communications and Communications Server Node Operator Facility”, on page 1 contains the following chapters:

- Chapter 1, “Introduction”, on page 3, describes the purpose of this book.
- Chapter 2, “Overview of the Verbs in This Book”, on page 7, describes the Node Operator Facility API structure and the verbs it supports. The chapter outlines the categories of the verbs implemented and the additional signals provided by Personal Communications and Communications Server.
- Chapter 3, “Node Operator Facility Entry Points”, on page 17, describes the entry point extensions.
- Chapters 4 through 12 describe the syntax of each verb. A copy of the structure that holds the information for each verb is included and each entry described, followed by a list of possible return codes.

Part 2, “Personal Communications and Communications Server Management Services API”, on page 595, contains the following chapters:

- Chapter 13, “Introduction to Management Services API”, on page 597, describes the management services API.
- Chapter 14, “Management Services Entry Points”, on page 601, describes the entry points for the management services verbs.
- Chapter 15, “Management Services Verbs”, on page 611, describes the syntax of each verb. A copy of the structure that holds the information for each verb is included and each entry described, followed by a list of possible return codes.

Icons

In this book, when it is necessary to communicate special information, the following icons appear:

This icon represents a note, important information that can affect the operation of Personal Communications or Communications Server or the completion of a task.

This icon appears when the information applies only to the Personal Communications program.

This icon appears when the information applies only to the Communications Server program.

Number Conventions

Binary numbers	Represented as BX'xxxx xxxx' or BX'x' except in certain instances where they are represented with text (“A value of binary xxxx xxxx is...”).
Bit positions	Start with 0 at the rightmost position (least significant bit).
Decimal numbers	Decimal numbers over 4 digits are represented in metric style. A space is used rather than a comma to separate groups of 3 digits. For example, the number sixteen thousand, one hundred forty-seven is written 16 147.

Hexadecimal numbers	Represented in text as hex xxxx or X'xxxx' (“The address of the adjacent node is hex 5D, which is specified as X'5d'.”)

Where to Find More Information

For more information, refer to *Quick Beginnings*, which contains a complete description of both the Communications Server library and related publications.

To view a specific book after Communications Server has been installed, use the following path from your desktop:

1. Programs
2. IBM Communications Server
3. Documentation
4. Choose from the list of books

The Communications Server books are in Portable Document Format (PDF), which is viewable with the Adobe Acrobat Reader. If you do not have a copy of this program on your machine, you can install it from the Documentation list.

The Communications Server home page on the Internet has general product information as well as service information about APARs and fixes. To get the home page, using an Internet browser, go to the following URL:

http://www.ibm.com/software/network/commserver/

For more information, refer to *Quick Beginnings*, which contains a complete description of both the Personal Communications library and related publications.

The Personal Communications books are included on the CD-ROM in portable document format (pdf). Books can be accessed directly from the root directory of the Personal Communications CD-ROM or from the Install Manager welcome panel.

To view the Personal Communications documentation using Install Manager, select **View Documentation** from the main panel of the Install Manager on the CD-ROM. Clicking **View Documentation** invokes Adobe Acrobat Reader from your system to view the books. If Acrobat Reader is not detected on your system, you are given the opportunity to install it at this time. After installation of Acrobat Reader is complete, a window opens displaying the books available on the CD-ROM.

Notes:

1. You can copy the book files from the CD-ROM to a local or network drive to view at a later time.
2. *Quick Beginnings* in HTML format is installed during installation of Personal Communications.

The Personal Communications home page on the Internet has general product information as well as service information about APARs and fixes. To get the home page, using an Internet browser such as IBM Web Explorer, go to the following URL:

http://www.ibm.com/software/network/pcomm/

The complete ***IBM Dictionary of Computing*** is available on the World Wide Web at http://www.ibm.com/networking/nsg/nsgmain.htm.

Part 1. Personal Communications and Communications Server Node Operator Facility

Chapter 1. Introduction

This part describes the Node Operator Facility (NOF) API provided by Personal Communications and Communications Server.

Purpose of the Document

The intent of this book is to:
- Provide a brief overview of the structure of the Node Operator Facility API
- Define the syntax of the signals that flow across the interface.

Personal Communications and Communications Server Node Operator Facility

The Personal Communications and Communications Server Node Operator Facility enables communication between the node operator, and the control point (CP) and logical units (LUs). The Node Operator Facility receives node configuration information from the operator, which it uses to initializethe control point when the node is started. The Node Operator Facility also receives requests to query and display node configuration information. The node operator is able to:
- Define and delete LUs, DLCs, ports, and links
- Activate and deactivate links and sessions
- Query the control point and LUs for database and status information

The node operator can be a human operator working with an interactive display, a command file accessed by a file interface, or a transaction program. The Node Operator Facility communicates with the node operator by using a verb interface.

Entry Points

Personal Communications and Communications Server provide a library file that handles Node Operator Facility verbs.

Node Operator Facility verbs have a straightforward language interface. Your program fills in fields in a block of memory called a *verb control block*. Then your program calls the entry point and passes a pointer to the verb control block. When its operation is complete, Node Operator Facility returns, having used and then modified the fields in the verb control block. Your program can then read the returned parameters from the verb control block.

Following is a list of entry points for Node Operator Facility verbs:
- WinNOF()
- WinAsyncNOF()
- WinAsyncNOFEx()
- WinNOFCancelAsyncRequest()
- WinNOFCleanup()
- WinNOFStartup()
- WinNOFRegisterIndicationSink()
- WinNOFUnregisterIndicationSink()
- WinNOFGetIndication()

See Chapter 3, “Node Operator Facility Entry Points”, on page 17 for detailed descriptions of the entry points.

Verb Control Blocks (VCBs)

Programming Note: The base operating system optimizes performance by executing some subsystems in the calling application’s address space. This means that incorrect use of local descriptor table (LDT) selectors by application programs can cause improper operation, or perhaps system failures.Accordingly, application programs should not perform pointer arithmetic operations that involve changing the LDT selector field of a pointer.

The segment used for the verb control block (VCB) must be a read/write data segment. Your program can either declare the VCB as a variable in your program, allocate it, or suballocate it from a larger segment. It must be sufficiently large to contain all the fields for the verb your program is issuing.

An application program should not change any part of the verb control block after it has been issued until the verb completes. When Node Operator Facility finishes the execution of a verb, it copies a complete, modified VCB back onto the original block. Therefore, if your program declares a verb control block as a variable, consider declaring it in static storage rather than on the stack of an internal procedure.

Fill all reserved and unused fields in each VCB with zeros (X'00'). In fact, it might be more time-efficient to set the entire verb control block to zeros before your program assigns the values to the parameters. Setting reserved fields to zeros is particularly important.

Note: If the VCB is not read/write, or if it is not at least 10 bytes (that is, large enough to hold the Node Operator Facility primary and secondary return codes), Node Operator Facility cannot access it, and the base operating system abnormally ends the process. This termination is recognized as a *general protection fault*, processor exception trap D.

Node Operator Facility returns the INVALID_VERB_SEGMENT primary return code when the VCB is too short or the incorrect type of segment is used.

Writing Node Operator Facility (NOF) Programs

Personal Communications and Communications Server provide a dynamic link library (DLL) file, that handles NOF verbs.

The DLL is reentrant; multiple application processes and threads can call the DLL concurrently.

NOF verbs have a straightforward language interface. Your program fills in fields in a block of memory called a *verb control block* (VCB). Then it calls the NOF DLL and passes a pointer to the verb control block. When its operation is complete, NOF returns, having used and then modified the fields in the VCB. Your program can then read the returned parameters from the verb control block.

Table 1 on page 5 shows source module usage of supplied header files and libraries needed to compile and link NOF programs. Some of the header files may include other required header files.

Table 1. Header Files and Libraries for NOF

Operating System	Header File	Library	DLL Name
WINNT & WIN95	WINNOF.H	WINNOF32.LIB	WINNOF32.DLL

Communications Server SNA API Client Support

This information applies only to Communications Server.

Included with Communications Server are a set of clients for the Windows 2000, Windows 95, Windows NT, and Windows XP operating systems. These clients are referred to as SNA API clients in this book and only support a subset of the full node operator facility. Specifically, **WINNOF** is the only API supported on the Windows 2000, Windows 95, Windows NT, and Windows XP clients. The following is a list of the NOF verbs supported:

- QUERY_LOCAL_LU
- QUERY_LU_0_TO_3
- QUERY_LU_POOL
- QUERY_MODE
- QUERY_MODE_DEFINITION
- QUERY_PARTNER_LU
- QUERY_PARTNER_LU_DEFINITION
- QUERY_PU
- QUERY_SESSION
- QUERY_TP
- QUERY_TP_DEFINITION

Verbs Supported by Communications Server and Not by Personal Communications

This information applies only to Communications Server.

The following list of verbs are supported by Communications Server and not by Personal Communications.

- DEFINE_DOWNSTREAM_LU
- DEFINE_DOWNSTREAM_LU_RANGE
- DEFINE_DSPU_TEMPLATE
- DELETE_DOWNSTREAM_LU
- DELETE_DOWNSTREAM_LU_RANGE
- DELETE_DSPU_TEMPLATE
- QUERY_ADJACENT_NN
- QUERY_DIRECTORY_STATS
- QUERY_DOWNSTREAM_LU

- QUERY_DOWNSTREAM_PU
- QUERY_DSPU_TEMPLATE
- QUERY_HPR_STATS
- QUERY_ISR_SESSION
- QUERY_NN_TOPOLOGY_NODE
- QUERY_NN_TOPOLOGY_STATS
- QUERY_NN_TOPOLOGY_TG
- DOWNSTREAM_LU_INDICATION
- DOWNSTREAM_PU_INDICATION
- ISR_INDICATION
- NN_TOPOLOGY_NODE_INDICATION
- NN_TOPOLOGY_TG_INDICATION

Chapter 2. Overview of the Verbs in This Book

The verb interface described in this book allows your programs to perform most of the configuration, system management, and node definition functions associated with a Personal Communications or Communications Server network environment. This chapter provides an overview of each of these functions and the associated verbs.

How to Read Verb Descriptions

Chapters 4 through 12 describe the configuration, system management, and attach manager verbs.

Supplied Parameters

Each verb description has a section that provides a detailed description of the parameters and any associated parameter values supplied by the program.

In some cases, you must supply a variable value for a parameter.

Returned Parameters

Each verb description has a section that provides a detailed description of the parameters and any associated parameter values returned to the program.

Return Codes

The configuration, system management, and attach manager verbs described in this book have return codes associated with them that supply information about the success of verb execution or that provide error information. These codes are listed in the "Returned Parameters" section for each verb.

Additional Information

Many of the verb descriptions also contain a section titled "Additional Information." This section provides additional useful information about the verb.

Common VCB Fields

This chapter documents the syntax of each verb passed across the Node Operator Facility API. It also describes the parameters passed in and returned for each verb.

```
typedef struct nof_hdr
{
        unsigned short  opcode;
        unsigned char   reserv2;             /* reserved */
        unsigned char   format;
        unsigned short  primary_rc;
        unsigned long   secondary_rc;
} NOF_HDR;
```

Each VCB has a number of common fields. These are listed and described below.

opcode
: Verb operation code. This field identifies the verb.

format
Identifies the format of the VCB. The value that this field must be set to in order to specify the current version of the VCB is documented individually under each verb.

primary_rc
Primary return code. Possible values for each verb are listed in each verb section.

secondary_rc
Secondary return code. This supplements the information provided by the primary return code.

Verb Summary

The Node Operator Facility API is composed of verbs that can be used to do the following things:
- Configure node resources
- Activate and deactivate links and sessions
- Query information held by the node
- Change the number of sessions
- Handle unsolicited indications
- Provide password support
- "ping" a remote LU
- Define, query, and delete CPI-C side information

Node Configuration

The following verbs can be used to define resources:
- DEFINE_ADJACENT_NODE
- DEFINE_CN
- DEFINE_COS
- DEFINE_DEFAULT_PU
- DEFINE_DLC
- DEFINE_DLUR_DEFAULTS
- DEFINE_DOWNSTREAM_LU

DEFINE_DOWNSTREAM_LU is Communications Server only.

- DEFINE_DOWNSTREAM_LU_RANGE

DEFINE_DOWNSTREAM_LU_RANGE is Communications Server only.

- DEFINE_DSPU_TEMPLATE
- DEFINE_FOCAL_POINT
- DEFINE_INTERNAL_PU
- DEFINE_LOCAL_LU
- DEFINE_LS
- DEFINE_LU62_TIMEOUT
- DEFINE_LU_0_TO_3
- DEFINE_LU_0_TO_3_RANGE

- DEFINE_LU_POOL
- DEFINE_MODE
- DEFINE_PARTNER_LU
- DEFINE_PORT
- DEFINE_TP

The following verbs can be used to delete resources:
- DELETE_ADJACENT_NODE
- DELETE_CN
- DELETE_COS
- DELETE_DLC
- DELETE_DOWNSTREAM_LU

DELETE_DOWNSTREAM_LU is Communications Server only.

- DELETE_DOWNSTREAM_LU_RANGE

DELETE_DOWNSTREAM_LU_RANGE is Communications Server only.

- DELETE_DSPU_TEMPLATE
- DELETE_FOCAL_POINT
- DELETE_INTERNAL_PU
- DELETE_LOCAL_LU
- DELETE_LS
- DELETE_LU62_TIMEOUT
- DELETE_LU_0_TO_3
- DELETE_LU_0_TO_3_RANGE
- DELETE_LU_POOL
- DELETE_MODE
- DELETE_PARTNER_LU
- DELETE_PORT
- DELETE_TP

Activation and Deactivation

The following verbs are used at link level:
- START_DLC
- START_LS
- START_PORT
- STOP_DLC
- STOP_LS
- STOP_PORT

The following verbs are used for dependent LU requestor function:
- START_INTERNAL_PU
- STOP_INTERNAL_PU

The following verbs are used at session level:

- ACTIVATE_SESSION
- DEACTIVATE_CONV_GROUP
- DEACTIVATE_SESSION

The following verb is used to force a high performance routing (HPR) RTP connection to switch paths:
  PATH_SWITCH

Querying the Node

These verbs return node information in named fields:

- QUERY_DEFAULT_PU
- QUERY_DLUR_DEFAULTS
- QUERY_MDS_STATISTICS
- QUERY_NN_TOPOLOGY_STATS

QUERY_NN_TOPOLOGY_STATS is Communications Server only.

- QUERY_NODE
- QUERY_STATISTICS

The following verbs can return one or more units of information:

- QUERY_ADJACENT_NN
- QUERY_ADJACENT_NODE
- QUERY_CN
- QUERY_CN_PORT
- QUERY_COS
- QUERY_DEFAULTS
- QUERY_DLUS
- QUERY_DOWNSTREAM_PU

QUERY_DOWNSTREAM_PU is Communications Server only.

- QUERY_DSPU_TEMPLATE
- QUERY_FOCAL_POINT
- QUERY_LU_POOL
- QUERY_LU62_TIMEOUT
- QUERY_MDS_APPLICATION
- QUERY_MODE_TO_COS_MAPPING
- QUERY_NMVT_APPLICATION
- QUERY_PU
- QUERY_TP

This information can be thought of as being stored in the form of a list. The verb can specify a named entry in the list, which is then considered to be a place marker (or index value) in the list. The **list_options** field on these verbs specifies from which point in the list information will be returned.

- If **list_options** is set to AP_FIRST_IN_LIST, then the fields specifying the index value will be ignored, and the returned list will start at the beginning.
- If **list_options** is set to AP_LIST_INCLUSIVE, then the returned list will start from the specified index value.
- If **list_options** is set to AP_LIST_FROM_NEXT, then the returned list will start from the entry after the specified index value.

The index value specifies the starting point for returned information. Once this has been determined, some of the query verbs also provide additional filtering options for the returned list. These are specified independently of the index value. Note that unless specified otherwise, the returned list will be ordered according to IBM's 6611 APPN® MIB. (See Appendix A, "IBM APPN MIB Tables", on page 627, for information on how verb parameters map to MIB table entries.)

The number of entries to be returned or the buffer size to be filled is set. (If both are set, then the verb is returned with the lower of the two specified quantities of information.) Because the application buffer size typically limits the amount of information that can be returned, the Node Operator Facility returns additional information indicating the total amount of buffer space required to return the requested information, and the total number of entries this represents.

In addition to returning one or more units of information, the following verbs are also able to return different levels of information. The **list_options** field specifies whether summary or detailed information will be returned by including either AP_DETAIL or AP_SUMMARY in the **list_options** field. These options are specified by **ORing** one of the previous **list_options**, for example: AP_DETAIL | AP_FIRST_IN_LIST.

- QUERY_DIRECTORY_LU
- QUERY_DLC
- QUERY_DLUR_LU
- QUERY_DLUR_PU
- QUERY_DOWNSTREAM_LU

QUERY_DOWNSTREAM_LU is Communications Server only.

- QUERY_ISR_SESSION

QUERY_ISR_SESSION is Communications Server only.

- QUERY_LOCAL_LU
- QUERY_LOCAL_TOPOLOGY
- QUERY_LS
- QUERY_LU_0_TO_3
- QUERY_MODE
- QUERY_MODE_DEFINITION
- QUERY_NN_TOPOLOGY_NODE

QUERY_NN_TOPOLOGY_NODE is Communications Server only.

- QUERY_NN_TOPOLOGY_TG

QUERY_NN_TOPOLOGY_TG is Communications Server only.

- QUERY_PARTNER_LU
- QUERY_PARTNER_LU_DEFINITION
- QUERY_PORT
- QUERY_RTP_CONNECTION
- QUERY_SESSION
- QUERY_TP_DEFINITION

Session Limit Verbs

- CHANGE_SESSION_LIMIT
- INITIALIZE_SESSION_LIMIT
- RESET_SESSION_LIMIT

Unsolicited Indications

Applications displaying node information can use these indications (which are issued when a change occurs and return summary information only) to trigger the query verbs (returning detailed information). The node only produces the signals listed below as unsolicited indications of the named events if there are any applications registered to receive the information. Applications should therefore unregister if they no longer require the information.

- DLC_INDICATION
- DLUR_LU_INDICATION
- DLUS_INDICATION
- DOWNSTREAM_LU_INDICATION

DOWNSTREAM_LU_INDICATION is Communications Server only.

- DOWNSTREAM_PU_INDICATION

DOWNSTREAM_PU_INDICATION is Communications Server only.

- FOCAL_POINT_INDICATION
- ISR_INDICATION

ISR_INDICATION is Communications Server only.

- LOCAL_LU_INDICATION
- LOCAL_TOPOLOGY_INDICATION

- LS_INDICATION
- LU_0_TO_3_INDICATION
- MODE_INDICATION
- NN_TOPOLOGY_NODE_INDICATION

NN_TOPOLOGY_NODE_INDICATION is Communications Server only.

- NN_TOPOLOGY_TG_INDICATION

NN_TOPOLOGY_TG_INDICATION is Communications Server only.

- PLU_INDICATION
- PORT_INDICATION
- PU_INDICATION
- REGISTRATION_FAILURE
- RTP_INDICATION
- SESSION_INDICATION
- SESSION_FAILURE_INDICATION

The entry points used for indications are:

WinNOFRegisterIndicationSink
Register to receive an indication

WinNOFUnregisterIndicationSink
Unregister from receiving an indication

WinNOFGetIndication
Receive an indication

These indications are passed to any indication sinks that have registered with the Node Operator Facility. If the component generating the indication is unable to send it, then it sets the **data_lost** indicator on the next indication it issues. If the **data_lost** flag has been set to AP_YES on an indication, then subsequent data fields can be set to null. This flag is used to notify the application that a change has occurred whose details have been lost, indicating that the application should respond by issuing the appropriate query verb.

Note that the signal LULU_EVENT is also classified as an indication as it is sent unsolicited by the node to a process registered using the verbs REGISTER_LULU_EVENT and UNREGISTER_LULU_EVENT. It is not listed above, since its behavior is significantly different: registration is for an LU-Partner LU pair, and there is no equivalent of **data_lost** — these LULU event indications are generated without fail.

Security Verbs

The following security verbs allow management of passwords for LU_LU verification or conversation security.
- DEFINE_LU_LU_PASSWORD
- DEFINE_USERID_PASSWORD
- DELETE_LU_LU_PASSWORD

- DELETE_USERID_PASSWORD

APING Verbs

The APING verb allows a management application to ping a remote LU in the network.

CPI-C Verbs

The following verbs allow CPI-C side information to be defined, queried, and deleted.

- DEFINE_CPIC_SIDE_INFO
- DELETE_CPIC_SIDE_INFO
- QUERY_CPIC_SIDE_INFO

Refer to *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications and Communications Server.

Attach Manager Verbs

The following verbs can be used to control the attach manager:

- DISABLE_ATTACH_MANAGER
- ENABLE_ATTACH_MANAGER
- QUERY_ATTACH_MANAGER

DLC Processes, Ports, and Link Stations

DLC Processes

Personal Communications or Communications Server can create multiple DLC processes. Each DLC process is created by Personal Communications or Communications Server in response to a START_DLC verb issued at the Node Operator Facility API. Each DLC is responsible for communication over a link, or set of links, using a specific data link protocol (such as SDLC or Token Ring).

Each DLC process can manage one or more ports. Ports are described below.

Ports

A port represents a unique access point (such as a MAC/SAP address pair) in the local machine and is associated with a DLC process. Each DLC can have one or more ports. A port can be one of the following types:

Switched port
Can have one or more adjacent link stations that are active at any one time. (Note that this differs from the definition in the *SNA APPN Architecture Reference*.)

Nonswitched port
Can have both point-to-point and multipoint link connections. Adjacent link stations on a nonswitched link connection must be defined by a Node Operator Facility component. Multipoint nonswitched links require primary/secondary relationships to be defined properly on all nodes to avoid unpredictable results.

SATF port
Uses a shared-access transport facility such as token ring. It allows connectivity between any pair of link stations attaching to the facility. The

initial role for all link stations being activated on a token ring must always be defined as negotiable, so that link activation can be initiated through any link station.

Note: SATF ports can also be associated with Connection Networks. In this case, topology updates are used to broadcast the address of the unique access point.

Link Stations

A link station is associated with a port and represents a connection to an adjacent node. A port can have multiple link stations. Link stations can be categorized in the following way:

Defined link station
A link station that has been defined explicitly (using a DEFINE_LS verb).

Dynamic link station
A link station that has been created as a result of activating a dynamic connection through a connection network (also known as a virtual routing node (VRN)).

Implicit link station
A link station that has been created as a result of a call received from a previously unknown partner node on a switched or SATF port. (This type of port is not defined in the *SNA APPN Architecture Reference* .)

Temporary link station
A link station that is created when a CONNECT_IN is received over the DLC interface on a switched or SATF port. It is either deleted, or becomes dynamic or implicit, when the remote node identity is determined.

Chapter 3. Node Operator Facility Entry Points

This chapter describes the entry points for Node Operator Facility verbs.

WinNOF()

This function provides a synchronous entry point for all of the Node Operator Facility verbs.

Syntax

```
void WINAPI WinNOF(long vcb,unsigned short vcb_size)
```

Parameters

vcb Pointer to verb control block.

vcb_size
Number of bytes in the verb control block.

Returns

No return value. The **primary_rc** and **secondary_rc** fields in the verb control block indicate any error.

Remarks

This is the main synchronous entry point for the Node Operator Facility API. This call blocks until the verb completes.

WinAsyncNOF()

This function provides an asynchronous entry point for all of the Node Operator Facility verbs.

Syntax

```
HANDLE WINAPI WinAsyncNOF(HWND hWnd,
                          long vcb,
                          unsigned short vcb_size)
```

Parameters

hWnd Window handle to receive completion message.

vcb Pointer to verb control block.

vcb_size
Number of bytes in the verb control block.

Returns

The return value specifies whether the asynchronous request was successful. If the function was successful, the actual return value is a handle. If the function was not successful, a zero is returned.

Remarks

Each application thread can only have one outstanding request at a time when using this entry point.

When the asynchronous operation is complete, the application's window *hWnd* receives the message returned **RegisterWindowMessage** with **"WinAsyncNOF"** as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call.

If the function returns successfully, a **WinAsyncNOF()** message will be posted to the application when the operation completes or the conversation is canceled.

Note: See also **WinNOFCancelAsyncRequest()** on page 21.

WinAsyncNOFEx()

This function provides an asynchronous entry point for all of the Node Operator Facility verbs. Use this entry point instead of the blocking calls to allow multiple verbs to be handled on the same thread.

Syntax

```
HANDLE WINAPI WinAsyncNOFEx(HANDLE handle,
                            long vcb,
                            unsigned short vcb_size);
```

Parameters

handle
Handle of the event that the application will wait on.

vcb Pointer to verb control block.

vcb_size
Number of bytes in the verb control block.

Returns

The return value specifies whether the asynchronous request was successful. If the function was successful, the actual return value is a handle.

Remarks

This entry point is intended for use with WaitForMultipleObjects in the Win32 API. For more information about this function, see the programming documentation for the Win32 API.

When the asynchronous operation is complete, the application is notified by way of the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

Note: See also **WinNOFCancelAsyncRequest()** on page 21.

WinNOFCancelAsyncRequest()

This function cancels an outstanding **WinAsyncNOF** based request.

Syntax

```
int WINAPI WinNOFCancelAsyncRequest(HANDLE handle);
```

Parameters

handle
Supplied parameter; specifies the handle of the request to be canceled.

Returns

The return value specifies whether the asynchronous request was canceled. If the value is zero, the request was canceled. Otherwise the value is:

WNOFALREADY
An error code indicating that the asynchronous request being canceled has already completed, or the handle was not valid.

Remarks

An asynchronous request previously issued by one of the **WinAsyncNOF** functions can be canceled prior to completion by issuing the **WinNOFCancelAsyncRequest()** call, specifying the handle returned by the initial function in *handle*.

Canceling an asynchronous request stops any update to the application verb control block and stops the application being notified that the verb has completed (either by way of the window message or event). It does not cancel the underlying request. To actually cancel the underlying request, the application must issue the appropriate NOF verb (that is, STOP_LS to cancel START_LS).

Should an attempt to cancel an existing asynchronous **WinAsyncNOF** routine fail with an error code of WNOFALREADY, one of two things has occurred. Either the original routine has already completed and the application has dealt with the resulting notification, or the original routine has already completed but the application has not dealt with the completion notification.

Note: See also **WinAsyncNOF()** on page 19.

WinNOFCleanup()

This function terminates and unregisters an application from the Node Operator Facility API.

Syntax

```
BOOL WINAPI WinNOFCleanup(void);
```

Returns

The return value specifies whether the unregistration was successful. If the value is not zero, the application was successfully unregistered. The application was not unregistered if a value of zero is returned.

Remarks

Use **WinNOFCleanup()** to indicate unregistration of a Node Operator Facility application from the Node Operator Facility API.

WinNOFCleanup unblocks any thread waiting in **WinNOFGetIndication**. These return with WNOFNOTREG, (the application is not registered to receive indication). **WinNOFCleanup** unregisters the application for all indications. **WinNOFCleanup** returns any outstanding verb (synchronous or asynchronous) with the error AP_CANCELLED. However, the verb completes inside the node.

It is not a requirement to use **WinNOFStartup** and **WinNOFCleanup**. However, an application must be consistent in its use of these calls. You should use both of them or never use either of them.

Note: See also **WinNOFStartup()** on page 23.

WinNOFStartup()

This function allows an application to specify the version of Node Operator Facility API required and to retrieve the version of the API supported by the product. This function can be called by an application before issuing any further Node Operator Facility API calls to register itself.

Syntax

```
int WINAPI WinNOFStartup(WORD wVersionRequired,
                        LPWNOFDATA nofdata);
```

Parameters

wVersionRequired
: Specifies the version of Node Operator Facility API support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.

nofdata
: Returns the version of Node Operator Facility API and a description of API implementation.

Returns

The return value specifies whether the application was registered successfully and whether the Node Operator Facility API implementation can support the specified version number. If the value is zero, it was registered successfully and the specified version can be supported. Otherwise, the return value is one of the following values:

WNOFSYSERROR
The underlying network subsystem is not ready for network communication.

WNOFVERNOTSUPPORTED
The version of Node Operator Facility API support requested is not provided by this particular implementation.

WNOFBADPOINTER
Incorrect nofdata parameter.

Remarks

This call is intended to help with compatibility of future releases of the API. The current version is 1.0.

It is not a requirement to use **WinNOFStartup** and **WinNOFCleanup**. However, an application must be consistent in its use of these calls. You should use both of them or never use either of them.

Note: See also **WinNOFCleanup()** on page 22.

WinNOFRegisterIndicationSink()

This allows the application to register to receive unsolicited indications.

Syntax

```
BOOL WINAPI WinNOFRegisterIndicationSink(unsigned short indication_opcode,
                                          unsigned short queue_size,
                                          unsigned short *primary_rc,
                                          unsigned long *secondary_rc);
```

Parameters

indication_opcode
The indication to register for.

queue_size
Number of unreceived indications to queue. Zero means use the current value (the initial default value is set to 10). There is only one queue for all indications registered by application.

primary_rc
Returned: primary return code

secondary_rc
Returned: secondary return code

Returns

The function returns a value indicating whether the registration was successful. If the value is not zero, the registration was successful. If the value is zero, the registration was not successful.

Remarks

Use **WinNOFRegisterIndicationSink** to register to receive unsolicited indications of type **indication_opcode**.

An application must issue a **WinNOFRegisterIndicationSink** for each type of indication it wants to receive.

Note: See also **WinNOFUnregisterIndicationSink()** on page 25 and **WinNOFGetIndication()** on page 26.

WinNOFUnregisterIndicationSink()

This allows the application to stop receiving unsolicited indications.

Syntax

```
BOOL WINAPI WinNOFUnregisterIndicationSink(unsigned short indication_opcode,
                                           unsigned short *primary_rc,
                                           unsigned long *secondary_rc);
```

Parameters

indication_opcode
The indication to unregister from.

primary_rc
Returned: primary return code.

secondary_rc
Returned: secondary return code.

Returns

The function returns a value indicating whether the unregistration was successful. If the value is not zero, the unregistration was successful. If the value is zero, the unregistration was not successful.

Remarks

Use **WinNOFUnregisterIndicationSink** to stop receiving unsolicited indications of type **indication_opcode**.

An application must issue a **WinNOFUnregisterIndicationSink** for each type of indication it wants to stop receiving.

Note: See also **WinNOFRegisterIndicationSink()** on page 24 and **WinNOFGetIndication()** on page 26.

WinNOFGetIndication()

This allows the application to received unsolicited indications.

Syntax

```
int WINAPI WinNOFGetIndication(long buffer,
                               unsigned short *buffer_size,
                               unsigned long timeout);
```

Parameters

buffer Pointer to a buffer to receive indication.

buffer_size
Size of buffer. Returned: the size of the indication.

timeout
Time to wait for indication in milliseconds.

Returns

The function returns a value indicating whether an indication was received.

0 Indication returned.

WNOFTIMEOUT
Timeout waiting for indication.

WNOFSYSNOTREADY
The underlying network subsystem is not ready for network communication.

WNOFNOTREG
The application is not registered to receive indications.

WNOFBADSIZE
The buffer is too small to receive the indication. Reissue the **WinNOFGetIndication** call with a large enough buffer. The size of the indication is returned in the **buffer_size** parameter.

WNOFBADPOINTER
Either the buffer or **buffer_size** parameter is not valid.

WNOFSYSERROR
An unexpected system error has occurred.

Remarks

This is a blocking call, it returns in one of the following circumstances:
- An indication is returned
- The timeout expires
- The application issues a WinNOFCleanup call
- The product is stopped
- A system error occurs

Note: See also **WinNOFRegisterIndicationSink()** on page 24 and **WinNOFUnregisterIndicationSink()** on page 25.

Chapter 4. Node Configuration Verbs

The following verbs are used to define and delete node configuration information.

DEFINE_ADJACENT_NODE

DEFINE_ADJACENT_NODE adds entries to the node directory database for the resources on an adjacent node.

Note: This verb is not required, and should not be issued, if there is an active path to the adjacent node using CP-CP sessions.

This verb can be issued on an end node, in which case the node's control point is added to the root of the directory.

To define the node's control point LU, set the following fields:
- Specify the node's control point name in the **cp_name** field
- Add an ADJACENT_NODE_LU structure, specifying the control point name in the **fqlu_name** field.

Any additional LUs on the node are added to the directory as children of the node's control point.DEFINE_ADJACENT_NODE can also be used to add LU definitions to an existing node definition. LUs can be removed in the same way by issuing the DELETE_ADJACENT_NODE verb. If the verb fails part way through processing, all new directory entries are removed, leaving the directory as it was before the verb was issued.

VCB Structure

The DEFINE_ADJACENT_NODE verb contains a variable number of ADJACENT_NODE_LU overlays. The ADJACENT_NODE_LU structures are concatenated onto the end of DEFINE_ADJACENT_NODE structure.

```
typedef struct define_adjacent_node
{
        unsigned short  opcode;              /* verb operation code     */
        unsigned char   reserv2;             /* reserved                */
        unsigned char   format;              /* format                  */
        unsigned short  primary_rc;          /* primary return code     */
        unsigned long   secondary_rc;        /* secondary return code   */
        unsigned char   cp_name[17];         /* CP name                 */
        unsigned char   description[RD_LEN]; /* resource description    */
        unsigned char   reserv3[19];         /* reserved                */
        unsigned short  num_of_lus;          /* number of LUs           */
} DEFINE_ADJACENT_NODE;
typedef struct adjacent_node_lu
{
        unsigned char   wildcard_lu;         /* wildcard LU name        */
                                             /* indicator               */
        unsigned char   fqlu_name[17];       /* fully qualified LU name */
        unsigned char   reserv1[6];          /* reserved                */
} ADJACENT_NODE_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_ADJACENT_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

cp_name
: The fully qualified name of the control point in the adjacent end node. This should match the name the node sends on its XIDs (if it supports them), and the adjacent control point name specified on the DEFINE_LS for the link to the node. The name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

description
: Resource description (returned on QUERY_DIRECTORY_LU). This is a 16-byte (nonzero) string in a locally displayable character set. All 16 bytes are significant.

num_of_lus
: The number of adjacent LU overlays that follow the DEFINE_ADJACENT_NODE VCB.

adjacent_node_lu.wildcard_lu
: Indicates whether the specified LU name is a wildcard name (AP_YES or AP_NO).

adjacent_node_lu.fqlu_name
: The LU name to be defined. If this name is not fully qualified the network ID of the CP name is assumed. The name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of either one or two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

  When **wildcard_lu** is TRUE, a dot (.) followed by EBCDIC spaces means a Full Wildcard (that will match anything). All EBCDIC spaces will match anything beginning with the Net id of the CP Name.

Returned Parameters

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

primary_rc
: AP_OK

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

primary_rc
: AP_PARAMETER_CHECK

secondary_rc
: AP_INVALID_CP_NAME

  AP_INVALID_LU_NAME
  AP_INVALID_WILDCARD_NAME

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

primary_rc
: AP_STATE_CHECK

secondary_rc
: AP_INVALID_CP_NAME

AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

secondary_rc
AP_MEMORY_SHORTAGE

AP_DIRECTORY_FULL

DEFINE_CN

DEFINE_CN defines a connection network (also 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. It also provides 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 that access the connection network by issuing another DEFINE_CN. (Ports can be removed in the same way by issuing the DELETE_CN verb.)

VCB Structure

```
typedef struct define_cn
{
        unsigned short  opcode;          /* verb operation code       */
        unsigned char   attributes;      /* verb attributes           */
        unsigned char   reserv2;         /* reserved                  */
        unsigned char   format;          /* format                    */
        unsigned short  primary_rc;      /* primary return code       */
        unsigned long   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[RD_LEN];
                                         /* resource description      */
        unsigned char   num_ports;       /* number of ports on CN     */
        unsigned char   reserv1[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;     /* modem class               */
        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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

fqcn_name
Fully qualified name (17 bytes long) of connection network being defined. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

def_data.description
Resource description (returned on QUERY_CN). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.num_ports
Number of ports associated with this connection network. There can be as many as eight ports per DEFINE_CN verb, and up to and including 239 ports in total per CN.

def_data.tg_chars.effect_cap
Actual units of effective capacity. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 2 eeeee, where the bit representation of the byte is `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 values as described in the list below:

AP_SEC_NONSECURE
No security exists.

AP_SEC_PUBLIC_SWITCHED_NETWORK
Data transmitted over this connection network will flow over a public switched network.

AP_SEC_UNDERGROUND_CABLE
Data transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT
The line is a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT
Conduit is protected against physical tapping.

AP_SEC_ENCRYPTED
Encryption over the line.

AP_SEC_GUARDED_RADIATION
Line is protected against physical and radiation tapping.

def_data.tg_chars.prop_delay
Propagation delay representing the time it takes for a signal to travel the

length of the link, in microseconds. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 2 eeeee, where the bit representation of the byte is `eeeeemmm`. Default values are listed below:

AP_PROP_DELAY_MINIMUM
No propagation delay.

AP_PROP_DELAY_LAN
Less than 480 microseconds delay.

AP_PROP_DELAY_TELEPHONE
Between 480 and 49 512 microseconds delay.

AP_PROP_DELAY_PKT_SWITCHED_NET
Between 49 512 and 245 760 microseconds delay.

AP_PROP_DELAY_SATELLITE
Longer than 245 760 microseconds delay.

AP_PROP_DELAY_MAXIMUM
Maximum propagation delay.

def_data.tg_chars.modem_class
Reserved. This field should always be set to zero.

def_data.tg_chars.user_def_parm_1
User defined parameter in the range 0–255.

def_data.tg_chars.user_def_parm_2
User defined parameter in the range 0–255.

def_data.tg_chars.user_def_parm_3
User defined parameter in the range 0–255.

port_name
Array of up to eight port names defined on the connection network. Each named port must have already been defined by a DEFINE_PORT verb. Each port name is an 8-byte string in a locally displayable character set and must match that on the associated DEFINE_PORT verb. Additional ports can be defined on the connection network by issuing another DEFINE_CN specifying the new port names.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CN_NAME

AP_INVALID_NUM_PORTS_SPECIFIED
AP_INVALID_PORT_NAME

AP_INVALID_PORT_TYPE
AP_DEF_LINK_INVALID_SECURITY
AP_EXCEEDS_MAX_ALLOWED

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PORT_ACTIVE

AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_COS

DEFINE_COS adds a class-of-service definition. The DEFINE_COS verb can also be used to modify any fields in a previously defined COS.

The definition provides node and TG rows. These rows associate a range of node and TG characteristics with weights that are 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** overlays. The **cos_tg_row** structures are concatenated onto the end of DEFINE_COS (and ordered in ascending weight) and are followed by the **cos_node_row** structures (also ordered in ascending weight).

```
typedef struct define_cos
{
        unsigned short  opcode;           /* verb operation code         */
        unsigned char   reserv2;          /* reserved                    */
        unsigned char   format;           /* format                      */
        unsigned short  primary_rc;       /* primary return code         */
        unsigned long   secondary_rc;     /* secondary return code       */
        unsigned char   cos_name[8];      /* class-of-service name       */
        unsigned char   description[RD_LEN];
                                          /* resource description        */
        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_node_row
{
        COS_NODE_STATUS minimum;          /* minimum                     */
        COS_NODE_STATUS maximum;          /* max                         */
        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;
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;      /* modem class                 */
        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_COS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

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

description
Resource description (returned on QUERY_COS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

transmission_priority
Transmission priority. This is set to one of the following values:

AP_LOW
AP_MEDIUM
AP_HIGH
AP_NETWORK

num_of_node_rows
Number of node row overlays that follow the DEFINE_COS VCB. The maximum is 8. 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 confines 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. Note that the node rows must be concatenated in ascending order of weight.

num_of_tg_rows
Number of TG row overlays that follow the node row overlays. 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 confines 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. Note that the TG rows must be concatenated 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. This can be one of the following values:

AP_UNCONGESTED
The node is not congested.

AP_CONGESTED
The number of ISR sessions is greater than the **isr_sessions_upper_threshold**.

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. This can be one of the following values:

AP_UNCONGESTED
The node is not congested.

AP_CONGESTED
The number of ISR sessions is greater than the **isr_sessions_upper_threshold**.

cos_node_row.weight
Weight associated with this node row. Values must be in the range 0–255.

cos_tg_row.minimum.effect_cap
Minimum limit for actual units of effective capacity. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 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.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 limits for security values as described in the list below:

AP_SEC_NONSECURE
No security exists.

AP_SEC_PUBLIC_SWITCHED_NETWORK
Data transmitted over this connection network will flow over a public switched network.

AP_SEC_UNDERGROUND_CABLE
Data transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT
The line is a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT
Conduit is protected against physical tapping.

AP_SEC_ENCRYPTED
Encryption over the line.

AP_SEC_GUARDED_RADIATION
Line is protected against physical and radiation tapping.

cos_tg_row.minimum.prop_delay
Minimum limits for propagation delay representing the time it takes for a signal to travel the length of the link, in microseconds. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 2 eeeee, where the bit representation of the byte is `eeeeemmm`. Default values are listed below:

AP_PROP_DELAY_MINIMUM
No propagation delay.

AP_PROP_DELAY_LAN
Less than 480 microseconds delay.

AP_PROP_DELAY_TELEPHONE
Between 480 and 49 512 microseconds delay.

AP_PROP_DELAY_PKT_SWITCHED_NET
Between 49 512 and 245 760 microseconds delay.

AP_PROP_DELAY_SATELLITE
Longer than 245 760 microseconds delay.

AP_PROP_DELAY_MAXIMUM
Maximum propagation delay.

cos_tg_row.minimum.modem_class
Reserved. This field should always be set to zero.

cos_tg_row.minimum.user_def_parm_1
Minimum limit for user-defined parameter in the range 0–255.

cos_tg_row.minimum.user_def_parm_2
Minimum limit for user-defined parameter in the range 0–255.

cos_tg_row.minimum.user_def_parm_3
Minimum limit for user-defined parameter in the range 0–255.

cos_tg_row.maximum.effect_cap
Maximum limit for actual units of effective capacity. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 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.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 limits for security values as described in the list below:

AP_SEC_NONSECURE
No security exists.

AP_SEC_PUBLIC_SWITCHED_NETWORK
Data transmitted over this connection network will flow over a public switched network.

AP_SEC_UNDERGROUND_CABLE
Data transmitted over secure underground cable.

AP_SEC_SECURE_CONDUIT
The line is a secure conduit that is not guarded.

AP_SEC_GUARDED_CONDUIT
Conduit that is protected against physical tapping.

AP_SEC_ENCRYPTED
Encryption over the line.

AP_SEC_GUARDED_RADIATION
Line is protected against physical and radiation tapping.

cos_tg_row.maximum.prop_delay
Maximum limits for propagation delay representing the time it takes for a signal to travel the length of the link, in microseconds. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 2 eeeee, where the bit representation of the byte is `eeeeemmm`. Default values are listed below:

AP_PROP_DELAY_MINIMUM
No propagation delay.

AP_PROP_DELAY_LAN
Less than 480 microseconds delay.

AP_PROP_DELAY_TELEPHONE
Between 480 and 49 512 microseconds delay.

AP_PROP_DELAY_PKT_SWITCHED_NET
Between 49 512 and 245 760 microseconds delay.

AP_PROP_DELAY_SATELLITE
Longer than 245 760 microseconds delay.

AP_PROP_DELAY_MAXIMUM
Maximum propagation delay.

cos_tg_row.maximum.modem_class
Reserved. This field should always be set to zero.

cos_tg_row.maximum.user_def_parm_1
Maximum limit for user-defined parameter in the range 0–255.

cos_tg_row.maximum.user_def_parm_2
Maximum limit for user-defined parameter in the range 0–255.

cos_tg_row.maximum.user_def_parm_3
Maximum limit for user-defined parameter in the range 0–255.

cos_tg_row.weight
Weight associated with this TG row.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_COS_NAME

AP_INVALID_NUMBER_OF_NODE_ROWS
AP_INVALID_NUMBER_OF_TG_ROWS
AP_NODE_ROW_WGT_LESS_THAN_LAST
AP_TG_ROW_WGT_LESS_THAN_LAST

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_COS_TABLE_FULL

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DEFAULT_PU

DEFINE_DEFAULT_PU allows the user to define, redefine, or modify any field of a default PU. It also allows the user to delete the default PU, by specifying a null PU name. If a PU name is not specified explicitly on a TRANSFER_MS_DATA verb, then the management services information carried on the TRANSFER_MS_DATA is sent on the default PU's session with the host SSCP. For more information about this see Chapter 15, "Management Services Verbs", on page 611.

VCB Structure

```
typedef struct define_default_pu
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   secondary_rc;   /* secondary return code */
        unsigned char   pu_name[8];     /* PU name               */
        unsigned char   description[RD_LEN];
                                        /* resource description  */
        unsigned char   reserv3[16];    /* reserved              */
} DEFINE_DEFAULT_PU;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_DEFAULT_PU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
Name of local PU that will serve as the default. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
Resource description (returned on QUERY_DEFAULT_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

Returned Parameters

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

primary_rc
AP_OK

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DEFAULTS

DEFINE_DEFAULTS allows the user to define or redefine default actions of the node.

VCB Structure

```
typedef struct define_defaults
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   secondary_rc;   /* secondary return code */
        DEFAULT_CHARS   default_chars;  /* default information   */
} DEFINE_DEFAULTS;
typedef struct default_chars
{
        unsigned char   description[RD_LEN];
                                        /* resource description  */
        unsigned char   mode_name[8];   /* default mode name     */
        unsigned char   implicit_plu_forbidden;
                                        /* disallow implicit     */
                                        /*  PLUs?                */
        unsigned char   specific_security_codes;
                                        /* generiuc security     */
                                        /* sense codes           */
        unsigned short  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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

default_chars.description
Resource description (returned on QUERY_DEFAULTS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

default_chars.mode_name
Name of the mode that will serve as the default. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

default_chars.implicit_plu_forbidden
Controls whether the Program puts implicit definitions in place for unknown Partner LUs (AP_YES or AP_NO).

default_chars.specific_security_codes
Controls whether the Program uses specific sense codes on a security authentication or authorization failure (AP_YES or AP_NO). Note, specific sense codes will only be returned to those partner LUs that have reported support for them on the session.

default_chars.limited_timeout
Specifies the timeout after which free limited-resource conwinner sessions will be deactivated. Range 0 to 65535 seconds.

Returned Parameters

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

primary_rc
AP_OK

If the verb specifies a default mode that is not valid (for example, not EBCDIC A), or is valid but has not been defined, the Program returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MODE_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

The effect of redefinition of each field is as follows:

description
The redefinition takes effect immediately. The updated description is returned on subsequent QUERY_DEFAULT verbs.

mode_name
The effect of a redefinition applies to all subsequent implicit mode definitions, for example, the updated mode serves as the default mode. The effect of a redefinition on a previously unknown mode (for example, one that had inherited the previous default mode characteristics) is identical to a redefinition of the unknown mode. For example, if the default mode is #INTER, and the Program receives a bIND for (an unknown) MODE1, the effect on MODE1 of the default mode subsequently being redefined to #BATCH should be identical to the effect of a DEFINE_MODE(MODE1) specifying the mode characteristics of #BATCH.

implicit_plu_forbidden
The redefinition takes effect immediately. All subsequent implicit PLU definitions succeed or fail depending on the updated value of this field.

specific_security_codes
The redefinition takes effect immediately.

limited_timeout
The updated value is used for all new session established after the redefinition. The old value is used for existing sessions.

DEFINE_DLC

DEFINE_DLC defines a new DLC or modifies an existing DLC. This verb defines the DLC name, which is unique throughout the node, and some DLC-specific data, which is concatenated to the basic structure. This data is used during initialization of the DLC, and the format is specific to the DLC type (such as Token Ring). Only the DLC-specific data appended to the verb can be modified using the DEFINE_DLC verb. To do this, a STOP_DLC verb must first be issued so that the DLC is in a reset state.

See "DLC Processes, Ports, and Link Stations" on page 14, for more information about the relationship between DLCs, ports and link stations.

VCB Structure

```
typedef struct define_dlc
{
        unsigned short  opcode;           /* verb operation code        */
        unsigned char   attributes;       /* verb attributes            */
        unsigned char   reserv2;          /* reserved                   */
        unsigned char   format;           /* format                     */
        unsigned short  primary_rc;       /* primary return code        */
        unsigned long   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
{
        DESCRIPTION     description;      /* resource description       */
        unsigned char   dlc_type;         /* DLC type                   */
        unsigned char   neg_ls_supp;      /* negotiable LS support      */
        unsigned char   port_types;       /* allowable port types       */
        unsigned char   hpr_only;         /* DLC only supports HPR links:*/
        unsigned char   reserv3;          /* reserved                   */
        unsigned char   retry_flags;      /* conditions for automatic   */
                                          /* retries                    */
        unsigned short  max_activation_attempts;
                                          /* how many automatic retries? */
        unsigned short  activation_delay_timer;
                                          /* delay between automatic    */
                                          /* retries                    */
        unsigned char   reserv4[4];          /* reserved                */
        unsigned short  dlc_spec_data_len; /* Length of DLC specific data */
} DLC_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_DLC

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dlc_name
Name of the DLC. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. For OEM devices, this name is manufacturer-specific. Valid values are LAN, SDLC, AnyNet®, X25 or TWINAX (padded to 8 chars with spaces).

def_data.description
Resource description (returned on QUERY_DLC). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.dlc_type
Type of the DLC.Personal Communications and Communications Server support the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

For EEDLC, use AP_OEM_DLC.

def_data.neg_ls_supp
Specifies whether the DLC supports negotiable link stations (AP_YES or AP_NO). If the **dlc_type** is AP_TWINAX, then only AP_NO is supported. If the **dlc_type** is AP_ANYNET, then only AP_YES is supported.

def_data.port_types
Specifies the allowable port types for the supplied **dlc_type**. The value corresponds to one or more of the following values ORed together.

AP_PORT_NONSWITCHED
AP_PORT_SWITCHED
AP_PORT_SATF

Use the following table to set the fields for the corresponding DLC type.

Table 2. Port Types for DLC Types

DLC Type	Port Type
AP_ANYNET	AP_PORT_SATF
AP_LLC2	AP_PORT_SATF
AP_OEM_DLC	AP_PORT_SWITCHED or AP_PORT_NONSWITCHED
AP_SDLC	AP_PORT_SWITCHED or AP_PORT_NONSWITCHED
AP_TWINAX	AP_PORT_NONSWITCHED
AP_X25	AP_PORT_SWITCHED or AP_PORT_NONSWITCHED

def_data.hpr_only
This field specifies whether the DLC only supports HPR links. This must be set to AP_YES for HPR over IP links.

AP_YES
AP_NO

def_data.retry_flags
This field specifies the conditions under which link stations are subject to automatic retry. It is a bit field, and may take any of the following values bit-wise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, the Program will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, the Program attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_INHERIT_RETRY
This flag has no effect.

def_data.max_activation_attempts
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags**, **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS, and **def_data.max_activation_attempts** on DEFINE_PORT is set to AP_USE_DEFAULTS.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means no limit. The value AP_USE_DEFAULTS means no limit.

def_data.activation_delay_timer
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags**, **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS, and **def_data.max_activation_attempts** on DEFINE_PORT is set to AP_USE_DEFAULTS.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **def_data.retry_flags**.

The value of zero or AP_USE_DEFAULTS results in the use of default timer duration of thirty seconds.

def_data.dlc_spec_data_len
This field should always be set to zero.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLC_NAME

AP_INVALID_DLC_TYPE
AP_INVALID_RETRY_FLAGS
AP_INVALID_PORT_TYPE
AP_HPR_NOT_SUPPORTED

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_DLC_ACTIVE

AP_INVALID_DLC_TYPE
AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DLUR_DEFAULTS

DEFINE_DLUR_DEFAULTS allows the user to define, redefine, or revoke a default dependent LU server (DLUS) and a backup default DLUS. 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. If a DLUS name is not specified explicitly on the DEFINE_DLUR_DEFAULTS verb then the current default (or backup DLUS) is revoked.

VCB Structure

```
typedef struct define_dlur_defaults
{
        unsigned short  opcode;             /* verb operation code   */
        unsigned char   reserv2;            /* reserved              */
        unsigned char   format;             /* format                */
        unsigned short  primary_rc;         /* primary return code   */
        unsigned long   secondary_rc;       /* secondary return code */
        unsigned char   description[RD_LEN];
                                            /* resource description  */
        unsigned char   dlus_name[17];      /* DLUS name             */
        unsigned char   bkup_dlus_name[17]; /* Backup DLUS name      */
        unsigned char   reserv3;            /* reserved              */
        unsigned short  dlus_retry_timeout; /* DLUS Retry Timeout    */
        unsigned short  dlus_retry_limit;   /* DLUS Retry Limit      */
        unsigned char   reserv4[16];        /* reserved              */
} DEFINE_DLUR_DEFAULTS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_DLUR_DEFAULTS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

description
Resource description. This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

dlus_name
Name of the DLUS node that will serve as the default. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If this field is set to all zeros, the current default DLUS is revoked.

bkup_dlus_name
Name of the DLUS node that will serve as the backup default. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If this field is set to all zeros, the current backup default DLUS is revoked.

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. If zero is specified, the default value of 5 seconds is used.

dlus_retry_limit
Maximum number of retries after an initial failure to contact a DLUS. If zero is specified, the default value of 3 is used. If X'FFFF' is specified, Personal Communications or Communications Server will retry indefinitely.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLUS_NAME

AP_INVALID_BKUP_DLUS_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DOWNSTREAM_LU

This verb applies only to Communications Server.

The DEFINE_DOWNSTREAM_LU verb is used for PU concentration. When PU concentration is used, downstream LUs are able to communicate with an upstream host. To do this, Communications Server maps each downstream LU to a dependent local LU, referred to as the *host LU*.

DEFINE_DOWNSTREAM_LU defines a new downstream LU and cannot be used to modify an existing definition. The downstream LU is mapped to the specified host LU (defined using the DEFINE_LU_0_TO_3 verb). The host LU can also be specified in terms of an LU pool.

When DEFINE_DOWNSTREAM_LU is issued for an existing downstream LU definition, the definition must be identical. If the downstream link is active and the downstream LU is inactive, the verb will be returned as successful and a reactivation attempt is made (although this may not be successful). If the downstream is not active or the downstream LU is already active, the verb failed. The processing of the reactivation attempt depends on the state of the specified host LU.

- If the host LU is active, then the ACTLU is resent to the downstream LU immediately.
- If the host LU is inactive, the node waits for the host LU to become active before sending the ACTLU to the downstream LU. The node attempts to activate the link to the host if it is not active (this will not be successful if the host link cannot be activated automatically).

VCB Structure

```
typedef struct define_downstream_lu
{
        unsigned short  opcode;           /* verb operation code       */
        unsigned char   attributes;       /* verb attributes           */
        unsigned char   reserv2;          /* reserved                  */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* primary return code       */
        unsigned long   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[RD_LEN];
                                          /* resource description      */
        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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dslu_name
Name of the downstream LU that is being defined. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.description
Resource description (returned on QUERY_DOWNSTREAM_LU). The length of this field should be a multiple of four bytes, and not zero.

def_data.nau_address
Network addressable unit address of the DOWNSTREAM LU. This must be in the range 1–255.

def_data.dspu_name
Name of the DOWNSTREAM PU (as specified on the DEFINE_LS). This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.host_lu_name
Name of the host LU or host LU pool that the downstream LU is mapped to. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.allow_timeout
Specifies whether the Program is allowed to time-out host LUs used by this dowstream LU if the session is left inactive for the **timeout** period specified on the host LU definition (AP_YES or AP_NO).

def_data.delayed_logon
Specifies whether the Program should delay connecting the downstream LU to the host LU until the first data is received from the dowstream LU. Instead, a simulated logon screen is sent to the downstream LU (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DNST_LU_NAME

AP_INVALID_NAU_ADDRESS

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_TYPE
AP_PU_NOT_DEFINED
AP_LU_ALREADY_DEFINED
AP_LU_NAU_ADDR_ALREADY_DEFD
AP_INVALID_HOST_LU_NAME
AP_LU_NAME_POOL_NAME_CLASH
AP_PU_NOT_ACTIVE
AP_LU_ALREADY_ACTIVATING
AP_LU_DEACTIVATING
AP_LU_ALREADY_ACTIVE
AP_CANT_MODIFY_VISIBILITY
AP_INVALID_ALLOW_TIMEOUT
AP_INVALID_DELAYED_LOGON
AP_DELAYED_VERB_PENDING

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DOWNSTREAM_LU_RANGE

This verb applies only to Communications Server.

The DEFINE_DOWNSTREAM_LU_RANGE verb is used for PU concentration. When PU concentration is used, downstream LUs are able to communicate with an upstream host. To do this, Communications Server maps each downstream LU to a dependent local LU, referred to as the *host LU*.

DEFINE_DOWNSTREAM_LU_RANGE allows the definition of multiple downstream LUs within a specified NAU range (but cannot be used to modify an existing definition). The node operator provides a base name and an NAU range. The LU names are generated by combining the base name with the NAU addresses.

For example, a base name of LUNME combined with an NAU range of 1 to 4 would define the LUs LUNME001, LUNME002, LUNME003, and LUNME004. A base name of less than five non-pad characters results in LU names of less than eight non-pad characters. Communications Server then right-pads these to eight characters.

Each downstream LU is mapped to the specified host LU (defined using the DEFINE_LU_0_TO_3 verb).

VCB Structure

```
typedef struct define_downstream_lu_range
{
        unsigned short  opcode;          /* verb operation code       */
        unsigned char   attributes;      /* verb attributes           */
        unsigned char   reserv2;         /* reserved                  */
        unsigned char   format;          /* format                    */
        unsigned short  primary_rc;      /* primary return code       */
        unsigned long   secondary_rc;    /* secondary return code     */
        unsigned char   dslu_base_name[5];/* Downstream LU base name   */
        unsigned char   description[RD_LEN];
                                         /* resource description      */
        unsigned char   min_nau;         /* min NAU address in range  */
        unsigned char   max_nau;         /* max 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 the */
                                         /* 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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dslu_base_name
Base name for downstream LU name range. This is a 5-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three type-A EBCDIC numeric characters, representing the decimal value of the NAU address, for each LU in the NAU range.

description
Resource description (returned on QUERY_DOWNSTREAM_LU). The length of this field should be a multiple of four bytes, and not zero.

min_nau
Minimum NAU address in the range. This can be from 1 to 255 inclusive.

max_nau
Maximum NAU address in the range. This can be from 1 to 255 inclusive.

dspu_name
Name of the DOWNSTREAM PU (as specified on the DEFINE_LS). This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

host_lu_name
Name of the host LU or host LU pool that all the downstream LUs within the range are mapped to. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

allow_timeout
Specifies whether the Program is allowed to time-out host LUs used by this downstream LU if the session is left inactive for the **timeout** period specified on the host LU definition (AP_YES or AP_NO).

delayed_logon
Specifies whether the Program should delay connection of the downstream LU to the host LU until the first data is received from the downstream LU. Instead, a simulated logon screen will be sent to the downstream LU (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DNST_LU_NAME

AP_INVALID_NAU_ADDRESS
AP_INVALID_ALLOW_TIMEOUT
AP_INVALID_DELAYED_LOGON

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LU_NAME_POOL_NAME_CLASH

AP_LU_ALREADY_DEFINED
AP_INVALID_HOST_LU_NAME
AP_PU_NOT_DEFINED
AP_INVALID_PU_NAME
AP_INVALID_PU_TYPE
AP_LU_NAU_ADDR_ALREADY_DEFD
AP_CANT_MODIFY_VISIBILITY
AP_DELAYED_VERB_PENDING

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_DSPU_TEMPLATE

This verb applies only to Communications Server.

This verb is used for PU concentration. When PU concentration is used, downstream LUs are able to communicate with an upstream host. To do this, Communications Server maps each downstream LU to a dependent local LU, referred to as the host LU. DEFINE_DSPU_TEMPLATE defines a template for the downstream LUs which are present on a group of downstream workstations. This template is used to put in place definitions for the downstream LUs when a workstation connects into Communications Server over an implicit link (one not previously defined). These templates are referred to by the **implicit_dspu_template** field on the DEFINE_PORT verb. DEFINE_DSPU_TEMPLATE can either be used to define a new template or to modify an existing template (although the existing instances of the modified template is not affected).

VCB Structure

```
typedef struct define_dspu_template
{
        unsigned short  opcode;             /* verb operation code      */
        unsigned char   attributes;         /* verb attributes          */
        unsigned char   format;             /* format                   */
        unsigned short  primary_rc;         /* primary return code      */
        unsigned long   secondary_rc;       /* secondary return code    */
        unsigned char   template_name[8];   /* name of template         */
        unsigned char   description;        /* resource description     */
        unsigned char   modify_template;    /* Modify existing template? */
        unsigned char   reserv1[11];        /* reserved                 */
        unsigned short  max_instance;       /* Max active template      */
                                            /* instances                */
        unsigned short  num_of_dslu_templates;
                                            /* number of DSLU templates  */
} DEFINE_DSPU_TEMPLATE;
typedef struct dslu_template
{
        unsigned char   min_nau;            /* min NAU address in range  */
        unsigned char   max_nau;            /* max 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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

template_name
: Name of the DSPU template. (This corresponds to the name specified in the **implicit_dspu_template** field on PORT_DEF_DATA). This is an 8_byte string in a locally-displayable character set. All 8 bytes are significant and must be set.

description
: Resource description (returned on QUERY_DSPU_TEMPLATE). The length of this should be a multiple of four bytes, and nonzero.

modify_template
: Specifies whether this verb should add additional DSLU templates to an existing DSPU template or should replace an existing DSPU template (AP_MODIFY_DSPU_TEMPLATE or AP_REPLACE_DSPU_TEMPLATE).

  If modify template is set to AP_MODIFY_DSPU_TEMPLATE and the named DSPU template does not exist, then it will be created.

  If **modify_template** is set to AP_MODIFY_DSPU_TEMPLATE and the named DSPU template does not exist, then appended DSLU templates are added to the existing DSPU template.

  If **modify_template** is set to AP_REPLACE_DSPU_TEMPLATE, then a new template is created. This can be from 0 to 65535 inclusive, where 0 means no limit.

max_instance
: This is the maximum number of instances of the template which can be active concurrently. While this limit is reached, no new instances can be created. This can be from 0 to 65535 inclusive, where 0 means no limit.

num_of_dslu_templates
: The number of DSLU template overlays which follow the DEFINE_DSPU_TEMPLATE VCB. This can be from 0 to 255 inclusive.

dslu_template.min_nau
: Minimum NAU address in the range. This can be from 1 to 255 inclusive.

dslu_template.max_nau
: Maximum NAU address in the range. This can be from 1 to 255 inclusive.

dslu_template.allow_timeout
: Specifies whether the Program is allowed to time-out host LUs used by this downstream LU if the session is left inactive for the **timeout** period specified on the host LU definition (AP_YES or AP_NO).

dslu_template.delayed_logon
: Specifies whether the Program should delay 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 (AP_YES or AP_NO).

dslu_template.host_lu
: Name of the host LU or host LU pool that all the downstream LUs within the range will be mapped onto. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC Spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TEMPLATE_NAME

AP_INVALID_NAU_ADDRESS
AP_INVALID_NAU_RANGE
AP_CLASHING_NAU_RANGE
AP_INVALID_NUM_DSPU_TEMPLATES
AP_INVALID_ALLOW_TIMEOUT
AP_INVALID_DELAYED_LOGON
AP_INVALID_MODIFY_TEMPLATE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_HOST_LU_NAME

AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_FOCAL_POINT

Personal Communications or Communications Server can have a number of types of relationships with different focal points. The DEFINE_FOCAL_POINT verb defines a focal point with which Personal Communications or Communications Server has an implicit relationship (which can be of type primary or backup). These relationships, and the ways in which they can be established, are described below. Relationships between a management services focal point (FP) and a management services entry point (EP) for a given category are established when they exchange Management Services Capabilities messages. The following types of FP-EP relationships can be established.

- Explicit
  This relationship is established by an operator at the focal point assigning the entry point to its sphere of control. The focal point initiates exchange of Management Services Capabilities.
- Implicit (primary)
  The relationship is established when an operator at an entry point assigns the entry point to a specified focal point (for example, when the operator issues a DEFINE_FOCAL_POINT verb). The entry point initiates the Management Services Capabilities exchange.
- Implicit (backup)
  This relationship is established when an entry point loses either an explicit or implicit primary focal point. The entry point initiates Management Services Capabilities exchange. The identity of the backup focal point can be defined (using the DEFINE_FOCAL_POINT verb) or can be acquired via Management Services Capabilities exchange.
- Default
  This relationship is established when an FP acquires an EP without operator intervention. The FP initiates the MS Capabilities exchange. This relationship only applies to EPs that are NNs
- Domain
  This relationship is established when a serving network node (NN) informs the end node entry point of the identity of the focal point. Domain relationships are only valid in end nodes.
- Host
  This relationship does not involve Management Services Capabilities exchange and is established by the configuration of an SSCP-PU session from the entry point node to a host. It is the lowest precedence focal point relationship.

Each DEFINE_FOCAL_POINT verb can only be used to define an implicit focal point (which can be of type primary or backup). Each DEFINE_FOCAL_POINT verb is issued for a specific management services category. Within this category the DEFINE_FOCAL_POINT verb can be used to
- Define a focal point
- Replace a focal point (or backup focal point)
- Revoke the currently active focal point.

The fields on a DEFINE_FOCAL_POINT verb are used as follows.

The **ms_category** must always be filled in. The combination of the **fp_fqcp_name** and **ms_appl_name fields** specify the focal point (or backup focal point if the **backup** field is set to AP_YES) for the specified category.

If the verb is being issued to revoke the currently active focal point without providing a new one, the **fp_fqcp_name** and **ms_appl_name** fields should be set to all zeros. When a DEFINE_FOCAL_POINT verb defining or replacing a focal point is received, Personal Communications or Communications Server attempts to establish an implicit primary focal point relationship with the specified focal point by sending a Management Services Capabilities request. When Personal Communications or Communications Server receives a DEFINE_FOCAL_POINT verb revoking the currently active focal point, it sends a Management Services Capabilities revoke message to the focal point. It is recommended that the DELETE_FOCAL_POINT verb (specifying AP_ACTIVE) be used to revoke the currently active focal point.

VCB Structure

```
typedef struct define_focal_point
{
        unsigned short  opcode;           /* verb operation code         */
        unsigned char   reserv2;          /* reserved                    */
        unsigned char   format;           /* format                      */
        unsigned short  primary_rc;       /* primary return code         */
        unsigned long   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[RD_LEN];
                                          /* resource description        */
        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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

ms_category
Management services category. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services categories as described in SNA management services, or an 8-byte type 1134 EBCDIC installation-defined name.

fp_fqcp_name
Focal point's fully qualified control point name. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the focal point is being revoked, this field should be set to all zeros.

ms_appl_name
Focal point application name. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services applications as described in SNA Management

Services, or an 8-byte type 1134 EBCDIC installation-defined name. If the focal point is being revoked, this field should be set to all zeros.

description
Resource description (returned on QUERY_FOCAL_POINT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

backup
Specifies whether a backup focal point is being defined (AP_YES or AP_NO). This field is reserved if the currently active focal point is being revoked. It is recommended that the DELETE_FOCAL_POINT verb (specifying AP_ACTIVE) be used to revoke the currently active focal point.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_FP_NAME

AP_INVALID_CATEGORY_NAME

If the verb does not execute successfully, the Program returns the following parameters:

primary_rc
AP_REPLACED

AP_UNSUCCESSFUL

secondary_rc
AP_IMPLICIT_REQUEST_REJECTED

AP_IMPLICIT_REQUEST_FAILED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

The Program returns the following parameter if the verb does not execute because of a system error or because the Program failed to contact the focal point successfully:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_INTERNAL_PU

The DEFINE_INTERNAL_PU verb defines a DLUR-served local PU. This verb is not used to define a local PU which is directly attached to the host. See "DEFINE_LS" on page 74 for this purpose.

Note: The DEFINE_LS verb should be used to define the following:
- A downstream PU served by:
  - DLUR
  - PU concentration
- A local PU that is directly attached to the host

VCB Structure

```
typedef struct define_internal_pu
{
        unsigned short  opcode;             /* verb operation code   */
        unsigned char   attributes;         /* verb attributes       */
        unsigned char   format;             /* format                */
        unsigned short  primary_rc;         /* primary return code   */
        unsigned long   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[RD_LEN];
                                            /* resource description  */
        unsigned char   dlus_name[17];      /* DLUS name             */
        unsigned char   bkup_dlus_name[17]; /* backup DLUS name      */
        unsigned char   pu_id[4];           /* PU identifier         */
        unsigned short  dlus_retry_timeout; /* DLUS retry timeout    */
        unsigned short  dlus_retry_limit;   /* DLUS retry limit      */
        unsigned char   conventional_lu_compression;
                                            /* Data compression      */
                                            /* requested for con-    */
                                            /* ventional LU sessions */
        unsigned char   conventional_lu_cryptography;
                                            /* Cryptography required */
                                            /* for conventional LU   */
                                            /* sessions              */
        unsigned char   reserv2[2]  ;         /* reserved            */
} INTERNAL_PU_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_INTERNAL_PU

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
: Name of the internal PU that is being defined. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.description
: Resource description (returned on QUERY_DLUR_PU and QUERY_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.dlus_name
: Name of the DLUS node that DLUR will use when it initiates SSCP-PU activation. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, the global default DLUS (if it has been defined, using the DEFINE_DLUR_DEFAULTS verb) is used in DLUR-initiated SSCP-PU activation.

def_data.bkup_dlus_name
: Name of the DLUS node that will serve as the backup DLUS for this PU. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, the global backup default DLUS (if it has been defined by the DEFINE_DLUR_DEFAULTS verb) is used as the backup for this PU.

def_data.pu_id
: PU identifier. This a 4-byte hexadecimal string. Bits 0–11 are set to the Block number and bits 12–31 to the ID number that uniquely identifies the PU. This must match the **pu_id** configured at the host.

def_data.dlus_retry_timeout
: Interval in seconds between second and subsequent attempts to contact the DLUS specified in the **def_data.dlus_name** and **def_data.bkup_dlus_name** fields. The interval between the initial attempt and the first retry is always one second. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. This field is ignored if **def_data.dspu_services** is not set to AP_DLUR.

def_data.dlus_retry_limit
: Maximum number of retries after an initial failure to contact the DLUS specified in the **def_data.dlus_name** and **def_data.bkup_dlus_name** fields. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. If X’FFFF’ is specified, the Program retries indefinitely. This field is ignored if **def_data.dspu_services** is not set to AP_DLUR.

def_data.conventional_lu_compression
: Specifies whether data compression is requested for conventional LU sessions dependent on this PU.

AP_NO
The local node should not be compressing or decompressing data flowing on conventional LU sessions using this PU.

AP_YES
Data compression should be enabled for conventional LU sessions dependent on this PU if the host requests compression. If this value is set, but the node does not support compression (defined on the START_NODE verb) then the INTERNAL_PU is successfully defined but without compression support.

def_data.conventional_lu_cryptography
Note: This function applies only to Communications Server.

Specifies whether session level encryption is required for conventional LU sessions dependent on this PU.

AP_NONE
The local node should not be compressing or decompressing data flowing on conventional LU sessions using this PU.

AP_MANDATORY
Mandatory session level encryption is performed by APPN if an import key is available to the LU. Otherwise, it must be performed by the application that uses the LU (if this is PU Concentration, then it is performed by a downstream LU).

AP_OPTIONAL
This value allows the cryptography used to be driven by the host application on a per session basis. If the host request cryptography for a session is dependent on this PU, then the behaviour of the Program is the same for AP_MANDATORY. If the host does not request cryptography, then the behaviour is the same as AP_NONE.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_ID
AP_INVALID_DLUS_NAME
AP_INVALID_BKUP_DLUS_NAME
AP_INVALID_CLU_CRYPTOGRAPHY

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PU_ALREADY_DEFINED

AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_LOCAL_LU

The DEFINE_LOCAL_LU verb requests the definition of a local LU with the specified characteristics, or, if the LU already exists, the modification of the **attach_routing_data** characteristic of the LU. Note that if a DEFINE_LOCAL_LU is used to modify an existing definition then any parameter other than the **attach_routing_data** field will be ignored.

VCB Structure

Format 1

```
typedef struct define_local_lu
{
        unsigned short  opcode;           /* verb operation code       */
        unsigned char   reserv2;          /* reserved                  */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* primary return code       */
        unsigned long   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;      /* resource description      */
        unsigned char   lu_alias[8];      /* local LU alias            */
        unsigned char   nau_address;      /* NAU address               */
        unsigned char   syncpt_support;   /* is sync-point supported?  */
        unsigned short  lu_session_limit; /* LU session limit          */
        unsigned char   default_pool;     /* member of default_lu_pool */
        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 */
        unsigned char   attach_routing_data;
                                          /* routing data for          */
                                          /* incoming attaches         */
        unsigned char   lu_model;         /* LU model for SDDLU        */
        unsigned char   model_name[7];    /* LU model name             */
                                          /* for SDDLU                 */
        unsigned char   reserv4[16];      /* reserved                  */
} LOCAL_LU_DEF_DATA;
```

VCB Structure

Format 0

```
typedef struct define_local_lu
{
        unsigned short  opcode;           /* verb operation code       */
        unsigned char   reserv2;          /* reserved                  */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* primary return code       */
        unsigned long   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;      /* resource description      */
        unsigned char   lu_alias[8];      /* local LU alias            */
        unsigned char   nau_address;      /* NAU address               */
        unsigned char   syncpt_support;   /* is sync-point supported?   */
        unsigned short  lu_session_limit; /* LU session limit          */
        unsigned char   default_pool;     /* member of default_lu_pool  */
        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 */
        unsigned char   attach_routing_data;
                                          /* routing data for           */
                                          /* incoming attaches          */
} LOCAL_LU_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_LOCAL_LU

format
Identifies the format of the VCB. Set this field to zero or one to specify either format 0 or format 1 of the VCB listed above.

lu_name
Name of the local LU that is being defined. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.description
Resource description (returned on QUERY_LOCAL_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.lu_alias
Alias of the local LU to define. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

def_data.nau_address
Network addressable unit address of the LU, which must be in the range 0–255. A nonzero value implies the LU is a dependent LU. Zero implies the LU is an independent LU.

def_data.syncpt_support
This field should always be set to AP_NO unless a sync point manager is available for this LU.

def_data.lu_session_limit
Maximum number of sessions supported by the LU. Zero means no limit. If the LU is independent then this can be set to any value. If the LU is dependent then this must be set to 1.

def_data.default_pool
Set to AP_YES if the LU is a member of the dependent LU6.2 default pool, or if it is to be used as the default Local LU for independent LU 6.2 sessions.

def_data.pu_name
Name of the PU that this LU will use. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is only used by dependent LUs, and should be set to all binary zeros for independent LUs.

def_data.lu_attributes
Specifies further information about the LU. This field either takes the value AP_NONE, or one or more following options ORed together.

AP_DISABLE_PWSUB
Disable password substitution support for the local LU.

def_data.sscp_id
This specifies the ID of the SSCP permitted to activate this LU. It is a 6-byte binary field. This field is only used by dependent LUs, and should be set to all binary zeros for independent LUs or if the LU may be activated by any SSCP.

def_data.disable
Indicates whether the LOCAL LU should be disabled or enabled. The LU can be dynamically enabled or disabled by reissuing the DEFINE_LOCAL_LU with this parameter set as appropriate (AP_YES or AP_NO). When a disabled LU is enabled, the Program issues a NOTIFY (on-line). When an enabled LU is disabled, the Program issues a NOTIFY (off-line). If the LU is bound when it is disabled, then the Program issues an UNBIND followed by a NOTIFY (off-line).

def_data.attach_routing_data
Type of attach routing data.

AP_REGISTERED_OR_DEFAULT_ATTACH_MGR
Specifies that a DYNAMIC_LOAD_INDICATION resulting from an attach arriving for the transaction program (TP) at this local LU is sent to the attach manager that has registered to receive DLIs for this LU, or to the default attach manager if no attach manager has registered for this LU.

AP_REGISTERED_ATTACH_MGR_ONLY
Specifies that a DYNAMIC_LOAD_INDICATION resulting from an attach arriving for the transaction program (TP) at this local LU is sent only to the attach manager that has registered to receive DLIs for this LU. If no attach manager has registered for this LU, the attach is rejected.

def_data.lu_model
Model type and number of the LU. This field is only used by dependent LUs and should be set to AP_UNKNOWN for independent LUs. For dependent LUs, this is set to one of the following values:

AP_3270_DISPLAY_MODEL_2
AP_3270_DISPLAY_MODEL_3
AP_3270_DISPLAY_MODEL_4
AP_3270_DISPLAY_MODEL_5
AP_RJE_WKSTN
AP_PRINTER
AP_SCS_PRINTER
AP_UNKNOWN

For dependent LUs, if **model_name** is not set to all binary zeros, then this field is ignored. If a value other than AP_UNKNOWN is specified and the host system supports SDDLU (Self-Defining Dependent LU), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector will contain the machine type and model number corresponding to the value of this field. This field may be changed dynamically by reissuing the verb. Changes will not come into effect until after the LU is closed and deactivated.

def_data.model_name
Model name of the LU. This field is only used by dependent LUs and should be set to binary zeros for independent LUs. APPN checks that this field consists of the EBCDIC characters A–Z, 0–9 and @, #, and $.

If this field is not set to binary zeros and the host system supports SDDLU, the node generates an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector contains the name supplied in this field. The **def_data.model_name** can be changed dynamically by reissuing the verb. Changes will not come into effect until after the LU is closed and deactivated.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_MODEL

AP_INVALID_LU_NAME
AP_INVALID_NAU_ADDRESS
AP_INVALID_SESSION_LIMIT
AP_INVALID_DISABLE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PU_NOT_DEFINED

AP_INVALID_LU_NAME
AP_LU_ALREADY_DEFINED
AP_ALLOCATE_NOT_PENDING
AP_LU_ALIAS_ALREADY_USED
AP_PLU_ALIAS_ALREADY_USED
AP_PLU_ALIAS_CANT_BE_CHANGED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

secondary_rc
AP_MEMORY_SHORTAGE

DEFINE_LS

DEFINE_LS is used to define a new link station (LS) or modify an existing one. This verb provides the LS name, which is unique throughout the node, and the name of the port this LS should use. This port must already have been defined using a DEFINE_PORT verb. Link-specific data is concatenated to the basic structure. DEFINE_LS can only be used to modify one or more fields of an existing link station if the link station is in a reset state (after a STOP_LS has been issued), and the **port_name** specified on the DEFINE_LS has not changed since the previous definition of the LS.

See “DLC Processes, Ports, and Link Stations” on page 14, for more information about the relationship between DLCs, ports, and link stations.

The setting of a large number of the fields in **LS_DEF_DATA** depends on the value of the **adj_cp_type** field. There are eight values that **adj_cp_type** can take (which are described further in **def_data.adj_cp_type** on page 78), four of which are used for links to adjacent Type 2.1 (APPN) nodes:
- AP_NETWORK_NODE
- AP_END_NODE
- AP_APPN_NODE
- AP_BACK_LEVEL_LEN_NODE

and four of which are used for links carrying PU Type 2.0 traffic only:
- AP_HOST_XID3
- AP_HOST_XID0
- AP_DSPU_XID
- AP_DSPU_NOXID.

There are four types of APPN nodes, which are distinguished as follows
- An APPN network node includes the Network Name Control Vector (CV) in its XID3, supports parallel TGs, sets the networking capabilities bit in its XID3, and can support CP-CP sessions on a link.
- An APPN end node includes the Network Name CV in its XID3, supports parallel TGs, does not set the networking capabilities bit in its XID3, and can support CP-CP sessions on a link.
- An up-level node includes the Network Name CV in its XID3, can support parallel TGs, does not set the networking capabilities bit in its XID3, and does not support CP-CP sessions.
- A back-level node does not include the Network Name CV in its XID3, does not support parallel TGs, does not set the networking capabilities bit in its XID3, and does not support CP-CP sessions.

The following fields must be set for all links:

port_name
adj_cp_type
dest_address
auto_act_supp
disable_remote_act
limited_resource
link_deact_timer
ls_attributes
adj_node_id
local_node_id

target_pacing_count
max_send_btu_size
link_spec_data_len
ls_role

Other fields must be set as follows:

- If **adj_cp_type** is set to AP_NETWORK_NODE, AP_END_NODE, or AP_APPN_NODE the following fields must be set:
  adj_cp_name
  tg_number
  solicit_sscp_sessions
  dspu_services
  hpr_supported
  hpr_link_lvl_error
  default_nn_server
  cp_cp_sess_support
  use_default_tg_chars
  tg_chars
- If **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE the following fields must be set:
  adj_cp_name
  solicit_sscp_sessions
  dspu_services
  use_default_tg_chars
  tg_chars
- If a local PU is to use the link (**adj_cp_type** is set to AP_HOST_XID3 or AP_HOST_XID0, or **solicit_sscp_sessions** is set to AP_YES on a link to an APPN node) the following field must be set:
  pu_name
- If a downstream PU is to use the link and will be served by PU Concentration (**dspu_services** is set to AP_PU_CONCENTRATION) the following field must be set:
  dspu_name
- If a downstream PU is to use the link and will be served by DLUR (**dspu_services** is set to AP_DLUR) the following fields must be set:
  dspu_name
  dlus_name
  bkup_dlus_name

VCB Structure

```
typedef struct define_ls
{
        unsigned short  opcode;             /* verb operation code           */
        unsigned char   attributes;         /* verb attributes               */
        unsigned char   reserv2;            /* reserved                      */
        unsigned char   format;             /* current format is zero        */
        unsigned short  primary_rc;         /* primary return code           */
        unsigned long   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[RD_LEN];
                                            /* resource description          */
        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 flag */
        unsigned char   dspu_services;      /* Services provided for         */
                                            /* downstream PU                 */
        unsigned char   dspu_name[8];       /* Downstream PU name (reserved  */
                                            /* if dspu_services is set to    */
                                            /* AP_NONE or AP_DLUR)           */
        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;      /* does the link support HPR?    */
        unsigned char   hpr_link_lvl_error; /* does link use link-level      */
                                            /* error recovery for HPR frms?  */
        unsigned short  link_deact_timer;   /* HPR link deactivation timer   */
        unsigned char   reserv1;            /* reserved                      */
        unsigned char   default_nn_server;  /* Use as deflt 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            */
        unsigned short  target_pacing_count;/* target pacing count           */
        unsigned short  max_send_btu_size;  /* max send BTU size             */
        unsigned char   ls_role;            /* link station role to use      */
                                            /*  on this link                 */
        unsigned char   max_ifrm_rcvd;      /* max number of I-frames rcvd   */
        unsigned short  dlus_retry_timeout; /* DLUS retry timeout            */
        unsigned short  dlus_retry_limit;   /* DLUS retry limit              */
        unsigned char   conventional_lu_compression;
                                            /* Data compression requested for */
                                            /* conventional LU sessions      */
        unsigned char   conventional_lu_cryptography;
                                            /* Cryptography required for     */
                                            /* conventional LU sessions      */
        unsigned char   reserv3;            /* reserved                      */
        unsigned char   retry_flags;        /* conditions LU sessions        */
        unsigned short  max_activation_attempts;
                                            /* how many automatic retries:   */
        unsigned short  activation_delay_timer;
                                            /* delay between automatic retries*/
        unsigned char   branch_link_type;   /* branch link type              */
        unsigned char   adj_brn_cp_support; /* adjacent BrNN CP support      */
        unsigned char   reserv4[20];        /* reserved                      */
        unsigned short  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;        /* modem class                   */
        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
{
        unsigned short  length;             /* length                         */
        unsigned short reserve1;            /* reserved                       */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                            /* address                        */
} LINK_ADDRESS;
typedef struct link_spec_data
{
       unsigned char  link_data[SIZEOF_LINK_SPEC_DATA];

}  LINK_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_LS

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Setting the field **ls_name** to the special value “$ANYNET$” (an ASCII string) has the effect of informing the Node Operator Facility that this is the link station to which independent LU session traffic that is to be routed by the AnyNet DLC should be sent. A link station of this name must be defined on a port over the AnyNet DLC if AnyNet routing is required.

def_data.description
Resource description (returned on QUERY_LS, QUERY_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.port_name
Name of port associated with this link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This named port must have already been defined by a DEFINE_PORT verb.

def_data.adj_cp_name
Fully qualified 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for

links to APPN nodes and is otherwise ignored. For links to APPN nodes it can be set to all zeros unless the field **tg_number** is set to a number in the range one to 20 or the field **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE. If it is set to all zeros, it is not checked against the name received from the adjacent node during XID exchange.If it is not set to all zeros, it is checked against the name received from the adjacent node during XID exchange unless **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE (in which case it is used to identify the adjacent node).

def_data.adj_cp_type
Adjacent node type.

AP_NETWORK_NODE
Specifies that the node is an APPN network node.

AP_END_NODE
Specifies that the node is an APPN end node or an up-level node.

AP_APPN_NODE
Specifies that the node is an APPN network node, an APPN end node, or an up-level node. The node type will be learned during XID exchange.

AP_BACK_LEVEL_LEN_NODE
Specifies that the node is a back_level_len node; that is, it does not send the control point name in the XID. For a link using the AnyNet DLC supporting independent LU sessions, you must specify AP_BACK_LEVEL_LEN_NODE.

AP_HOST_XID3
Specifies that the node is a host and that Personal Communications or Communications Server responds to a polling XID from the node with a format 3 XID.

AP_HOST_XID0
Specifies that the node is a host and that Personal Communications or Communications Server responds to a polling XID from the node with a format 0 XID. For a link using the AnyNet DLC supporting dependent LU sessions, you must specify AP_HOST_XID0.

AP_DSPU_XID
Specifies that the node is a downstream PU and that Personal Communications or Communications Server includes XID exchange in link activation.

AP_DSPU_NOXID
Specifies that the node is a downstream PU and that Personal Communications or Communications Server does not include XID exchange in link activation.

Note: A link station to a VRN is always dynamic and is therefore not defined.

def_data.dest_address.length
Length of destination link station's address on adjacent node.

If **def_data.dest_address.length** is set to zero and this LS is associated with a port of type SATF, then the Program considers this link station to be a wild card link station. This will cause the Program to match LS to any incoming connection that is not matched by another defined link station.

def_data.dest_address.address
Link station's destination address on adjacent node. For a link using the AnyNet DLC, the <b>dest_address</b> specifies the adjacent node ID or adjacent control point name. If an adjacent node ID is specified, the length must be 4 and the address must contain the 4-byte hexadecimal node ID (1-byte block ID, 3-byte PU ID). If an adjacent control point name is specified, the length must be 17 and the address must contain the control point name in EBCDIC, padded with EBCDIC blanks.

def_data.auto_act_supp
Specifies whether the link can be activated automatically when required by a session. (AP_YES or AP_NO). If the link is not to an APPN node then this field can always be set to AP_YES and has no requirements on other parameters. If the link is to an APPN node, then this field cannot be set to AP_YES if the link also supports CP-CP sessions; and can only be set to AP_YES if a preassigned TG number is also defined for the link **tg_number** and is set to a value between one and 20). These requirements will always be met if **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE because **cp_cp_sess_support** and **tg_number** are ignored in this case).

def_data.tg_number
Preassigned TG number. This field is only relevant if the link is to an adjacent APPN node and is otherwise ignored. If **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE then it is also ignored and is assumed to be set to one. For links to adjacent APPN nodes this must be set in the range one to 20. This number is used to represent the link when the link is activated. Personal Communications or Communications Server will not accept any other number from the adjacent node during activation of this link. To avoid link-activation failure because of a mismatch of preassigned TG numbers, the same TG number must be defined by the adjacent node on the adjacent link station (if using preassigned TG numbers). If a preassigned TG number is defined then the **adj_cp_name** must also be defined (and cannot be set to all zeros) and the **adj_cp_type** must be set to AP_NETWORK_NODE or AP_END_NODE. If zero is entered the TG number is not preassigned and is negotiated when the link is activated.

def_data.limited_resource
Specifies whether this link station is to be deactivated when there are no sessions using the link. This is set to one of the following values:

AP_NO
The link is not a limited resource and will not be deactivated automatically.

AP_YES or AP_NO_SESSIONS
The link is a limited resource and will be deactivated automatically when no active sessions are using it. A limited resource link station can be configured for CP-CP session support. (This is done by setting this field to AP_YES and **cp_cp_sess_support** to AP_YES.) In this case, if CP-CP sessions are brought up over the link, Personal Communications or Communications Server will not treat the link as a limited resource (and will not bring the link down).

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. Note that link stations on a nonswitched port cannot be configured as limited resource.

Note that link stations on a nonswitched port cannot be configured as limited resource.

A limited resource link station may be configured for CP-CP session support. (This is done by setting this field to AP_YES and **cp_cp_sess_support** to AP_YES.) In this case, if CP-CP sessions are brought up over the link, Personal Communications or Communications Server will not retreat the link as a limited resource (and will not bring the link down). Note, this does not apply if this field is set to AP_INACTIVITY.

def_data.solicit_sscp_sessions
AP_YES requests the adjacent node to initiate sessions between the SSCP and the local control point and dependent LUs. (In this case the **pu_name** must be set.) AP_NO requests no sessions with the SSCP on this link. This field is only relevant if the link is to an APPN node and is otherwise ignored. If the adjacent node is defined to be a host (**adj_cp_type** is set to AP_HOST_XID3 or AP_HOST_XID0), then Personal Communications or Communications Server always requests the host to initiate sessions between the SSCP and the local control point and dependent LUs (and again the **pu_name** must be set).

This field can only be set to AP_YES on a link to an adjacent APPN node if **dspu_services** is set to AP_NONE. If this field is set to AP_YES and the DCL used by this LS is defined as hpr_only, then the DEFINE_LS is rejected with a parameter check and secondary return code of AP_INVALID_SOLICIT_SSCP_SESS.

def_data.pu_name
Name of local PU that will use this link if the adjacent node is defined to be a host or **solicit_sscp_sessions** is set to AP_YES on a link to an APPN node. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If the adjacent node is not defined to be a host, and is not defined as an APPN node with **solicit_sscp_sessions** set to AP_YES, this field is ignored.

def_data.disable_remote_act
Specifies whether remote activation of this link is supported (AP_YES or AP_NO).

def_data.dspu_services
Specifies the services that the local node provides to the downstream PU across this link. This is set to one of the following:

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. This setting is only valid if the local node is a Network Node.

AP_NONE
Local node will provide no services for this downstream PU.

The **dspu_name** must also be set if this field is set to AP_PU_CONCENTRATION or AP_DLUR.

This field must be set to AP_PU_CONCENTRATION or AP_DLUR if the adjacent node is defined as a downstream PU (that is, **adj_cp_type** is set to AP_DSPU_XID or AP_DSPU_NOXID). It can be set to AP_PU_CONCENTRATION or AP_DLUR on a link to an APPN node if **solicit_sscp_sessions** is set to AP_NO. This field is ignored if the adjacent node is defined as a host.

If this field is not set to AP_NONE and the DLC used by this LS is defined as hpr_only, then the DEFINE_LS is rejected with a parameter check and secondary return code of SP_INVALID_DSPU_SERVICES.

def_data.dspu_name
Name of the downstream PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

This field must be set if **dspu_services** is set to AP_PU_CONCENTRATION or AP_DLUR and is otherwise ignored.

def_data.dlus_name
Name of DLUS node which DLUR solicits SSCP services from when the link to the downstream node is activated. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, then the global default DLUS (if it has been defined using the DEFINE_DLUR_DEFAULTS verb) is solicited when the link is activated. If the **dlus_name** is set to zeros and there is no global default DLUS, then DLUR will not initiate SSCP contact when the link is activated. This field is ignored if **dspu_services** is not set to AP_DLUR.

def_data.bkup_dlus_name
Name of DLUS node which serves as the backup for the downstream PU. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, then the global backup default DLUS (if it has been defined by the DEFINE_DLUR_DEFAULTS verb) is used as the backup for this PU. This field is ignored if **dspu_services** is not set to AP_DLUR.

def_data.hpr_supported
Specifies whether HPR is supported on this link (AP_YES or AP_NO). This field is only relevant if the link is to an APPN node and is otherwise ignored. If it is not, setting this field to AP_YES results in the verb being rejected with a parameter check and a secondary return code of INVALID_NODE_TYPE_FOR_HPR.

def_data.hpr_link_lvl_error
Specifies whether HPR traffic should be sent on this link using link-level error recovery (AP_YES or AP_NO). This parameter is ignored if **hpr_supported** is set to AP_NO.

def_data.link_deact_timer
Limited resource link deactivation timer (in seconds).

If **limited_resource** is set to AP_INACTIVITY, then a link is automatically deactivated if no data traverses the link for the duration of this timer.

If zero is specified, the default value of 30 is used. Otherwise, the minimum value is 5. (If it is set any lower, the specified value will be ignored and 5 will be used.) This parameter is reserved if **limited_resource** is set to AP_NO.

def_data.default_nn_server
Specifies whether a link can be automatically activated by an end node to support CP-CP sessions to a network node server. (AP_YES or AP_NO). Note that the link must be defined to support CP-CP sessions for this field to take effect.

def_data.ls_attributes
Specifies further information about the adjacent node.

def_data.ls_attributes[0]
Host type.

AP_SNA
Standard SNA host.

AP_FNA
FNA (VTAM-F) host.

AP_HNA
HNA host.

def_data.ls_attributes[1]
This is a bit field. It may take the value AP_NO, or any of the following values bit-wise ORed together.

AP_SUPPRESS_CP_NAME
Network Name CV suppression option for a link to a back-level LEN node. If this bit is set, no Network Name CV is included in XID exchanges with the adjacent node. (This bit is ignored unless **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE or AP_HOST_XID3.)

AP_REACTIVATE_ON_FAILURE
If the link is active and then fails, Personal Communications or Communications Server will attempt to reactivate the link. If the reactivation attempt fails, the link will remain inactive.

AP_USE_PU_NAME_IN_XID_CVS
If the adjacent node is defined to be a host or **solicit_sscp_sessions** is set to AP_YES on a link to an APPN node, and the AP_SUPPRESS_CP_NAME bit is not set, then the fully qualified CP name in Network Name CVs sent on Format 3 XIDs is replaced by the name supplied in **def_data.pu_name**, fully qualified with the network ID of the CP.

def_data.adj_node_id
Node ID of adjacent node. This a 4-byte hexadecimal string. If **adj_cp_type** indicates the adjacent node is a T2.1 node, this field is ignored unless it is nonzero, and either the **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE or the adjacent node does not send a Network Name CV in its XID3. If **adj_cp_type** is set to AP_HOST_XID3 or AP_HOST_XID0, this field is always ignored. If **adj_cp_type** is set to AP_DSPU_XID and this field is nonzero, it is used to check the identity of the downstream PU. If **adj_cp_type** is set to AP_DSPU_NOXID, this field is either ignored (if **dspu_services** is AP_PU_CONCENTRATION) or used to identify the downstream PU to DLUS (if **dspu_services** is AP_DLUR).

def_data.local_node_id
Node ID sent in XIDs on this link station. This a 4-byte hexadecimal string. If this field is set to zero, the **node_id** will be used in XID exchanges. If this field is nonzero, it replaces the value for XID exchanges on this LS.

def_data.cp_cp_sess_support
Specifies whether CP-CP sessions are supported (AP_YES or AP_NO). This field is only relevant if the link is to an APPN node and is otherwise ignored. If **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE then it is also ignored and is assumed to be set to AP_NO.

def_data.use_default_tg_chars
Specifies whether the default TG characteristics supplied on the DEFINE_PORT verb should be used (AP_YES or AP_NO). If this is set to AP_YES then the **tg_chars** field will be ignored. This field is only relevant if the link is to an APPN node and is otherwise ignored.

def_data.tg_chars
TG characteristics (See "DEFINE_CN" on page 31). This field is only relevant if the link is to an APPN node and is otherwise ignored.

def_data.target_pacing_count
Numeric value between 1 and 32 767, inclusive, indicating the desired pacing window size for BINDs on this TG. The number is only significant when fixed bind pacing is being performed. Personal Communications or Communications Server does not currently use 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 than can be transmitted between a link station pair. If the link is not HPR-capable then this must be set to a value greater than or equal to 99. If the link is HPR-capable then this must be set to a value greater than or equal to 768.

def_data.ls_role
The link station role that this link station should assume. This can be any one of AP_LS_NEG, AP_LS_PRI or AP_LS_SEC to select a role of negotiable, primary or secondary. The field can also be set to AP_USE_PORT_DEFAULTS to select the value configured on the DEFINE_PORT verb. If the **dlc_type** is AP_TWINAX, then only AP_LS_SEC is supported. If **dlc_type** is AP_ANYNET (and **ls_name** is ″$ANYNET$″), then AP_LS_PRI is not supported.

def_data.max_ifrm_rcvd
The maximum number of I-frames that can be received by the XID sender before acknowledgment.

Range: 0–127

If zero is specified, the value of **max_ifrm_rcvd** from DEFINE_PORT is used as the default.

def_data.dlus_retry_timeout
Interval in seconds between second and subsequent attempts to contact the DLUS specified in the **def_data.dlus_name** and **def_data.bkup_dlus_name** fields. The interval between the initial attempt and the first retry is always one second. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. This field is ignored if **def_data.dspu_services** is not set to AP_DLUR.

def_data.dlus_retry_limit
Maximum number of retries after an initial failure to contact the DLUS specified in the **def_data.dlus_name** and **def_data.bkup_dlus_name** fields. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. If X'FFFF' is specified, APPN retries indefinitely. This field is ignored if **def_data.dspu_services** is not set to AP_DLUR.

def_data.conventional_lu_compression
Specifies whether data compression is requested for conventional LU sessions dependent on this PU. Note that this field is only valid for links carrying LU 0 to 3 traffic.

AP_NO
The local node should not be compressing or decompressing data flowing on conventional LU sessions using this PU.

AP_YES
Data compression should be enabled for conventional LU sessions dependent on this PU if the host requests compression. If this value is set, but the node does not support compression (defined on the START_NODE verb) then the link station is successfully defined but without compression support.

def_data.conventional_lu_cryptography
Specifies whether session level encryption is required for conventional LU sessions. This field only applies to links carrying conventional LU traffic.

AP_NONE
Session level encryption is not performed by the Program.

AP_MANDATORY
Mandatory session level encryption is performed by the Program if an import key is available to the LU. Otherwise, it must be performed by the application that uses the LU (if this is PU Concentration, then it is performed by a downstream LU).

AP_OPTIONAL
This value allows the cryptography used to be driven by the host application on a per session basis. If the host requests cryptography for a session on this LS, then the behavior of the Program is the same as AP_MANDATORY. If the host does not request cryptography, then the behaviour is as for AP_NONE.

def_data.retry_flags
This field specifies the conditions under which activation of this link station is subject to automatic retry. It is a bit field, and may take any of the following values bit-wise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, the Program will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, the Program attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_INHERIT_RETRY
In addition to the retry conditions specified by flags in this field, those specified in the **retry_flags** field of the underlying port definition will also be used.

def_data.max_activation_attempts
This field has no effect unless at least one flag is set in **retry_flags**.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means no limit. The value AP_USE_DEFAULTS results in the use of **max_activiation_attempts** supplied on DEFINE_PORT.

def_data.activation_delay_timer
This field has no effect unless at least one flag is set in **retry_flags**.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **def_data.retry_flags**.

The value AP_USE_DEFAULTS results in the use of **activiation_delay_timer** supplied on DEFINE_PORT.

If zero is specified, the Program uses a default timer duration of thirty seconds.

def_data.branch_link_type
BrNN only. This specifies whether a link is an uplink or a downlink. This field only applies if the **def_data.adj_cp_type** is set to AP_NETWORK_NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE.

AP_UPLINK
This link is an uplink.

AP_DOWNLINK
The link is a downlink.

If the field **adj_cp_type** is set to AP_NETWORK_NODE, then this field must be set to AP_UPLINK.

Other node types: This field is ignored.

def_data.adj_brnn_cp_support
BrNN only. This specifies whether the adjacent CP is allowable, is a requirement, or prohibited from being an NN(BrNN); for example, a BrNN showing an NN face. This field only applies if the field **adj_cp_type** is set

to AP_NETWORK_NODE or AP_APPN_NODE (and the node type learned during XID exchange is network node).

AP_BRNN_ALLOWED
The adjacent CP is allowed (but not required) to be an NN(BrNN).

AP_BRNN_REQUIRED
The adjacent CP is required to be an NN(BrNN).

AP_BRNN_PROHIBITED
The adjacent CP is not allowed to be an NN(BrNN).

If the field **adj_cp_type** is set to AP_NETWORK_NODE and the field **auto_act_supp** is set to AP_YES, then this field must be set to AP_BRNN_REQUIRED or AP_BRNN_PROHIBITED.

Other node types: This field is ignored.

def_data.link_spec_data_len
This field should always be set to zero.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_DEF_LINK_INVALID_SECURITY

AP_INVALID_CP_NAME
AP_INVALID_LIMITED_RESOURCE
AP_INVALID_LINK_NAME
AP_INVALID_LS_ROLE
AP_INVALID_NODE_TYPE
AP_INVALID_PORT_NAME
AP_INVALID_AUTO_ACT_SUPP
AP_INVALID_PU_NAME
AP_INVALID_SOLICIT_SSCP_SESS
AP_INVALID_DLUS_NAME
AP_INVALID_BKUP_DLUS_NAME
AP_INVALID_NODE_TYPE_FOR_HPR
AP_INVALID_TARGET_PACING_COUNT
AP_INVALID_BTU_SIZE
AP_HPR_NOT_SUPPORTED
AP_INVALID_TG_NUMBER
AP_MISSING_CP_NAME
AP_MISSING_CP_TYPE
AP_MISSING_TG_NUMBER
AP_PARALLEL_TGS_NOT_SUPPORTED
AP_INVALID_DLUS_RETRY_TIMEOUT
AP_INVALID_DLUS_RETRY_LIMIT
AP_INVALID_CLU_CRYPTOGRAPHY

AP_INVALID_RETRY_FLAGS
AP_BRNN_SUPPORT_MISSING
AP_INVALID_BRANCH_LINK_TYPE
AP_INVALID_BRNN_SUPPORT

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LOCAL_CP_NAME

AP_DEPENDENT_LU_SUPPORTED
AP_DUPLICATE_DEST_ADDR
AP_INVALID_NUM_LS_SPECIFIED
AP_LS_ACTIVE
AP_PU_ALREADY_DEFINED
AP_DSPU_SERVICES_NOT_SUPPORTED
AP_DUPLICATE_TG_NUMBER
AP_TG_NUMBER_IN_USE
AP_CANT_MODIFY_VISIBILITY
AP_INVALID_UPLINK
AP_INVALID_DPWNLINK

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_LU_0_TO_3

This verb defines an LU of type 0, 1, 2 or 3. It allows the LU to be added to an LU pool. If the pool does not already exist, it is added. This verb cannot be used to modify the **lu_model**, **model_name**, **priority**, **description**, and **appc_spec_def_data** of an existing definition, but no other fields may be modified.

Personal Communications or Communications Server supports implicit LU type 0, 1, 2 or 3 definition by ACTLU. Implicit definitions cannot be deleted, but are removed when the LU becomes inactive. To obtain information about implicit definitions, use QUERY_LU_0_TO_3 or register for LU_0_TO_3_INDICATIONs. An implicit LU definition can be redefined using DEFINE_LU_0_TO_3, provided **lu_name**, **pu_name**, and **nau_address** are correct, and **pool_name** is all zeros (the LU is then treated as if it had been configured by the operator in the first place).

VCB Structure

Format 1

```
typedef struct define_lu_0_to_3
{
        unsigned short  opcode;                /* verb operation code       */
        unsigned char   attributes;            /* verb attributes           */
        unsigned char   format;                /* format                    */
        unsigned short  primary_rc;            /* primary return code       */
        unsigned long   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            /* resource description      */
        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                  */
        unsigned char   sscp_id[6]             /* SSCP ID                   */
        unsigned short  timeout;               /* Timeout                   */
        unsigned char   app_spec_def_data[16]; /* Application Specified Data */
        unsigned char   model_name[7];         /* LU model name for DDDLU   */
        unsigned char   reserv3[17];           /* reserved                  */
} LU_0_TO_3_DEF_DATA;
```

VCB Structure

Format 0

```
typedef struct define_lu_0_to_3
{
        unsigned short  opcode;                /* verb operation code       */
        unsigned char   attributes;            /* attributes                */
        unsigned char   format;                /* format                    */
        unsigned short  primary_rc;            /* primary return code       */
        unsigned long   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            /* resource description      */
        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                   */
    unsigned char   sscp_id[6]             /* SSCP ID                    */
    unsigned short  timeout;               /* Timeout                    */
    unsigned char   app_spec_def_data[16]; /* Application Specified Data */
  } LU_0_TO_3_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_LU_0_TO_3

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero or one to specify one of the versions of the VCB listed above.

lu_name
Name of the local LU that is being defined. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.description
Resource description (returned on QUERY_LU_0_TO_3). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.nau_address
Network addressable unit address of the LU, which must be in the range 1–255.

def_data.pool_name
Name of LU pool to which this LU belongs.This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If the LU does not belong to a pool, this field is set to all binary zeros. If the pool does not currently exist, it is created.

def_data.pu_name
Name of the PU (as specified on the DEFINE_LS verb) that this LU will use. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

def_data.priority
LU priority when sending to the host. This is set to one of the following values:

AP_NETWORK
AP_HIGH
AP_MEDIUM
AP_LOW

def_data.lu_model
Model type and number of the LU. This is set to one of the following values:

AP_3270_DISPLAY_MODEL_2
AP_3270_DISPLAY_MODEL_3
AP_3270_DISPLAY_MODEL_4
AP_3270_DISPLAY_MODEL_5
AP_RJE_WKSTN
AP_PRINTER
AP_SCS_PRINTER
AP_UNKNOWN

Format 1 only, if **model_name** is not set to all binary zeros, then this field is ignored.

If a value other than AP_UNKNOWN is specified and the host system supports DDDLU (Dynamic Definition of Dependent LUs), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. For format 1 only, the PSID subvector contains the machine type and model number corresponding to the value of this field. This field may be changed dynamically by reissuing the verb. Changes will not come into effect until the LU is next closed and deactivated.

def_data.sscp_id
This field specifies the ID of the SSCP permitted to activate this LU. It is a 6–byte binary field. If the field is set to binary zeros, then the LU may be activated by any SSCP.

def_data.timeout
Timeout for 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 holds:

- 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.

def_data.app_spec_def_data
Application specified defined data. This field is not interpreted by Personal Communications or Communications Server, but is stored and subsequently returned on the QUERY_LU_0_TO_3 verb.

def_data.model_name
Personal Communications or Communications Server checks that this field consists of the EBCDIC characters A–Z, 0–9 and @, #, and $. If this field is not set to all binary zeros and the host system supports DDDLU (Dynamic Definition of Dependent LUs), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector will contain the name supplied in this field. This field may be changed dynamically by reissuing the verb. Changes will not come into effect until the LU is closed and deactivated.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_PU_NAME
AP_INVALID_PU_TYPE
AP_PU_NOT_DEFINED
AP_LU_ALREADY_DEFINED
AP_LU_NAU_ADDR_ALREADY_DEFD
AP_CANT_MODIFY_VISIBILITY

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_TYPE
AP_PU_NOT_DEFINED
AP_LU_NAME_POOL_NAME_CLASH
AP_LU_ALREADY_DEFINED
AP_LU_NAU_ADDR_ALREADY_DEFD

If the verb does not execute because the system has not been built with Dependent LU support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_LU_0_TO_3_RANGE

This verb allows the definition of multiple LUs within a specified NAU range. The node operator provides a base name and an NAU range. The LU names are generated by combining the base name with the NAU addresses. This verb cannot be used to modify existing definitions.

For example, a base name of LUNME combined with an NAU range of 1 to 4 would define the LUs LUNME001, LUNME002, LUNME003, and LUNME004. A base name of less than five non-pad characters results in LU names of less than eight non-pad characters. Personal Communications or Communications Server then right-pads these to eight characters.

VCB Structure

Format 1

```
typedef struct define_lu_0_to_3_range
{
        unsigned short  opcode;               /* verb operation code       */
        unsigned char   attributes;           /* verb attributes           */
        unsigned char   format;               /* format                    */
        unsigned short  primary_rc;           /* primary return code       */
        unsigned long   secondary_rc;         /* secondary return code     */
        unsigned char   base_name[5];         /* base name                 */
        unsigned char   reserv3;              /* reserved                  */
        unsigned char   description;          /* resource description      */
        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                  */
        unsigned char   sscp_id[6];          /* SSCP ID                   */
        unsigned short  timeout;              /* Timeout                   */
        unsigned char   app_spec_def_data[16]; /* application specified data */
        unsigned char   model_name[7];        /* LU model name for DDDLU   */
        unsigned char   name_attributes;     /* Attributes of base name   */
        unsigned char   base_number;         /* Base number for LU names  */
        unsigned char   reserv3[15];          /* reserved                  */
} DEFINE_LU_0_TO_3_RANGE;
```

VCB Structure

Format 0

```
typedef struct define_lu_0_to_3_range
{
        unsigned short  opcode;               /* verb operation code       */
        unsigned char   attributes;           /* verb attributes           */
        unsigned char   format;               /* format                    */
        unsigned short  primary_rc;           /* primary return code       */
        unsigned long   secondary_rc;         /* secondary return code     */
        unsigned char   base_name[5];         /* base name                 */
        unsigned char   reserv3;              /* reserved                  */
        unsigned char   description;          /* resource description      */
        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                  */
        unsigned char   sscp_id[6];          /* SSCP ID                    */
        unsigned short  timeout;              /* Timeout                    */
        unsigned char   app_spec_def_data;    /* application specified data */
} DEFINE_LU_0_TO_3_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_LU_0_TO_3_RANGE

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero or one to specify one of the versions of the VCB listed above.

base_name
Base LU name. This is an 5-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three type-A EBCDIC numeric characters, representing the decimal value of the NAU address, for each LU in the NAU range.

This is the field with no bits set in the field **name_attributes**. Setting bits changes the meaning of this field.

description
Resource description (returned on QUERY_LU_0_TO_3). The length of this field should be a multiple of four bytes, and not zero.

min_nau
Minimum NAU address in the range. This can be from 1 to 255 inclusive.

max_nau
Maximum NAU address in the range. This can be from 1 to 255 inclusive.

pool_name
Name of LU pool to which this LU belongs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If the LU does not belong to a pool, this field is set to all binary zeros.

pu_name
Name of the PU (as specified on the DEFINE_LS verb) that this LU uses. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

priority
LU priority when sending to the host. This is set to one of the following values:

AP_NETWORK
AP_HIGH
AP_MEDIUM
AP_LOW

lu_model

Model type and number of the LU. This is set to one of the following values:

AP_3270_DISPLAY_MODEL_2
AP_3270_DISPLAY_MODEL_3
AP_3270_DISPLAY_MODEL_4
AP_3270_DISPLAY_MODEL_5
AP_RJE_WKSTN
AP_PRINTER
AP_SCS_PRINTER
AP_UNKNOWN

Format 1 only, if **model_name** is not set to all binary zeros, then this field is ignored.

If a value other than AP_UNKNOWN is specified and the host system supports DDDLU (Dynamic Definition of Dependent LUs), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. For format 1 only, the PSID subvector contains the machine type and model number corresponding to the value of this field. This field may be changed dynamically by reissuing the verb. Changes will not come into effect until the LU is next closed and deactivated.

sscp_id

This field specifies the ID of the SSCP permitted to activate this LU. It is a 6–byte binary field. If the field is set to binary zeros, then the LU may be activated by any SSCP.

timeout

Timeout for 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 holds:

- 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.

model_name

Personal Communications or Communications Server checks that this field consists of the EBCDIC characters A–Z, 0–9 and @, #, and $. If this field is not set to all binary zeros and the host system supports SDDLU (Self-Defining Dependent LU), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector will contain the name supplied in this field.

name_attributes

This bit field modifies the interpretation and usage of the supplied **base_name**. This field may take the value of zero, or any or all of the following values bit-wise ORed together.

AP_USE_HEX_IN_NAME

If this bit is set, the interpretation of the **base_name** is modified as follows:

This is an 6-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. The base name is appended with two EBCDIC characters, representing the hexadecimal value of the NAU address, for each LU in the NAU range.

AP_USE_BASE_NUMBER
If this bit is set, the interpretation **base_name** is modified as follows:

This is an 5-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three EBCDIC numeric characters, representing the decimal index of the LU in the range, starting with **base_number** and ending with (**base_name + max_nau — min_nau**).

base_number
If the AP_USE_BASE_NUMBER bit is not set in **name_attributes**, this field is ignored. Otherwise, this field modifies the interpretation of **base_name** described previously. Legal values are from zero to (255 – **max_nau + min_nau**).

app_spec_def_data
Application specified defined data. This field is not interpreted by Personal Communications or Communications Server, but is stored and subsequently returned on the QUERY_LU_0_TO_3 verb (the same data is returned for each LU in the range).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_BASE_NUMBER

AP_INVALID_LU_MODEL
AP_INVALID_LU_NAME
AP_INVALID_NAME_ATTRIBUTES
AP_INVALID_NAU_ADDRESS
AP_INVALID_PRIORITY

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PU_NOT_DEFINED

AP_INVALID_PU_NAME
AP_INVALID_PU_TYPE
AP_LU_NAME_POOL_NAME_CLASH
AP_LU_ALREADY_DEFINED
AP_LU_NAU_ADDR_ALREADY_DEFD
AP_IMPLICIT_LU_DEFINED
AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because the system has not been built with dependent LU support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_LU_POOL

This verb is used to define an LU pool or to add LUs to an existing pool. The LUs that are to be added must already have been defined using either a DEFINE_LU_0_TO_3 verb or a DEFINE_LU_0_TO_3_RANGE verb. LUs can only belong to one LU pool at a time. If the specified LUs already belong to a pool, they are removed from the existing pool into the pool being defined. Up to 10 LUs can be added to a pool at a time, although there is no limit to the total number of LUs in a pool.

VCB Structure

```
typedef struct define_lu_pool
{
        unsigned short  opcode;               /* verb operation code        */
        unsigned char   attributes;           /* verb attributes            */
        unsigned char   format;               /* format                     */
        unsigned short  primary_rc;           /* primary return code        */
        unsigned long   secondary_rc;         /* secondary return code      */
        unsigned char   pool_name[8];         /* LU pool name               */
        unsigned char   description[RD_LEN];  /* resource description       */
        unsigned char   reserv3[4];           /* reserved                   */
        unsigned short  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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pool_name
Name of pool to which these LUs belong. This name is an 8-byte string, padded to the right with spaces. This can be either an EBCDIC string or a string in a locally displayable character set.

description
Resource description (returned on QUERY_LU_POOL). The length of this field should be a multiple of four bytes, and not zero.

num_lus
Number of LUs to add, in the range 0–10.

lu_names
Names of the LUs that are being added to the pool. Each name is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_NUM_LUS
AP_INVALID_POOL_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LU_NAME_POOL_NAME_CLASH

AP_INVALID_POOL_NAME

If the verb does not execute because the system has not been built with dependent LU support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_MODE

The DEFINE_MODE verb defines a set of networking characteristics to assign to a particular mode (or group of sessions). This verb can also be used to modify any fields on a previously defined mode. If the SNASVCMG mode is redefined, its **mode_name** and **cos_name** cannot be modified. The CPSVCMG mode cannot be redefined.

The DEFINE_MODE verb can also be used to define the default COS, which unknown modes will be mapped to. This is done by setting **mode_name** to all zeros. The default COS is initially #CONNECT.

Note: It is not necessary to define all the modes you want to use locally, though they must be defined at your network node and potentially, the partner node. If an ALLOCATE is issued specifying a mode that has not been defined, the node uses the characteristics for the model default mode specified on the DEFINE_DEFAULTS verb. If no such model has been specified, the characteristics of the blank mode are used for the model.

VCB Structure

```
typedef struct define_mode
{
        unsigned short  opcode;               /* verb operation code         */
        unsigned char   reserv2;              /* reserved                    */
        unsigned char   format;               /* format                      */
        unsigned short  primary_rc;           /* primary return code         */
        unsigned long   secondary_rc;         /* secondary return code       */
        unsigned char   mode_name[8];         /* mode name                   */
        unsigned short  reserv3;              /* reserved                    */
        MODE_CHARS      mode_chars;           /* mode characteristics        */
} DEFINE_MODE;
```

```
typedef struct mode_chars
{
        unsigned char   description[RD_LEN]
                                              /* resource description        */
        unsigned short  max_ru_size_upp;      /* max RU size upper bound     */
        unsigned char   receive_pacing_win;   /* receive pacing window       */
        unsigned char   default_ru_size;      /* default RU size to maximize */
                                              /* performance                 */
        unsigned short  max_neg_sess_lim;     /* max negotiable session limit */
        unsigned short  plu_mode_session_limit; /* LU-mode session limit       */
        unsigned short  min_conwin_src;       /* min source contention winner */
                                              /* sessions                    */
        unsigned char   cos_name[8];          /* class-of-service name       */
        unsigned char   cryptography;         /* cryptography                */
        unsigned char   compression;          /* compression                 */
        unsigned short  auto_act;             /* initial auto-activation count*/
        unsigned short  min_conloser_src;     /* min source contention loser  */
        unsigned short  max_ru_size_low       /* maximum RU size lower bound  */
        unsigned short  max_receive_pacing_win;
                                              /* maximum receive pacing window*/
        unsigned char   max_compress_lvl;     /* maximum compression level    */
        unsigned char   max_decompression_lvl; /* maximum decompression level  */
        unsigned char   comp_in_series;       /* support for LZ and RLE       */
        unsigned char   reserv4[24];          /* reserved                    */
} MODE_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_MODE

format
Identifies the format of the VCB. Set this field to zero or one to specify the version of the VCB listed above.

mode_name
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. If this is set to all zeros, the default COS is set to **mode_chars.cos_name**, and all other **mode_chars** fields are ignored.

mode_chars.description
Resource description (returned on QUERY_MODE_DEFINITION and QUERY_MODE). The length of this field should be a multiple of four bytes, and nonzero.

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. The range is 256–61440. This field is ignored if **default_ru_size** is set to AP_YES.

mode_chars.receive_pacing_win
Session pacing window for sessions in this mode. For fixed pacing, this value specifies the receive pacing window. For adaptive pacing, this value is used as a preferred minimum window size. Note that Personal Communications or Communications Server will always use adaptive pacing unless the adjacent node specifies that it does not support it. The range is 1–63. The value zero is not allowed.

mode_chars.default_ru_size
Specifies whether a default upper bound for the maximum RU size will be used. If this parameter specifies AP_YES, **max_ru_size_upp** is ignored, and the upper bound for the maximum RU size is set to the link BTU size minus the size of the TH and the RH.

AP_YES
AP_NO

mode_chars.max_neg_sess_lim
Maximum number of sessions allowed on this mode between any local LU and partner LU. If a value of zero is specified then there will be no implicit CNOS exchange. The range is 0–32 767.

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. If a value of zero is specified then there will be no implicit CNOS exchange. The range is 0–32 767.

mode_chars.min_conwin_src
Minimum number of contention winner sessions that can be activated by any one local LU using this mode. This value is used when CNOS (Change Number of Sessions) exchange is initiated implicitly. If a value of zero is specified then there will be no implicit CNOS exchange. The range is 0–32 767.

mode_chars.cos_name
: Name of the class of service to request when activating sessions on this mode. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

mode_chars.cryptography
: Specifies whether session-level cryptography must be used (AP_NONE or AP_MANDATORY).

mode_chars.compression
: Specifies the use of compression for sessions activated using this mode.

  AP_COMP_PROHIBITED
  : Compression is not supported on sessions for this mode.

  AP_COMP_REQUESTED
  : Compression is supported and requested (but not mandated) on sessions for this mode.

    If the **format** field is set to 0, then the compression and decompression levels are set to the maximum supported by the node.

    If the **format** field is set to 1, then the maximum levels of compression and decompression are defined by the **max_compress_lvl** and **max_decompress_lvl** fields.

mode_chars.auto_act
: Specifies how many sessions are automatically activated for this mode. This value is used when Change Number of Sessions (CNOS) exchange is initiated implicitly.

  The range is 0–32767.

mode_chars.min_consloser_src
: Specifies the minimum number of contention loser sessions to be activated by any one local LU for this mode. This value is used when CNOS (change number of sessions) exchange is initiated implicitly. The range is 0–32767.

mode_chars.max_ru_size_low
: Specifies the lower bound for the maximum size of RUs sent and received on sessions in this mode. This value is used when the maximum RU size is negotiated during session activation. The range is 256–61140.

  The value zero means that there is no lower bound.

  The field is ignored if **default_ru_size** is set to AP_YES.

mode_chars.max_receive_pacing_win
: Specifies the maximum pacing window for sessions in this mode. For adaptive pacing, this value is used to limit the receive pacing window it grants. For fixed pacing, this field is not used. Note, the Program always uses adaptive pacing unless the adjacent node specifies that it does not support it. The range is 0–32767.

  The value of zero means that there is no upper bound.

mode_chars.max_compress_lvl
: The maximum compression level that the Program attempts to negotiate for data flowing supported by the node.

  AP_NONE
  AP_RLE_COMPRESSION

AP_LZ9_COMPRESSION
AP_LZ10_COMPRESSION
AP_LZ12_COMPRESSION

The level of compression configured cannot be greater than that supported by the node (specified in the field **max_compress_lvl** on START_NODE). Note, if compression is negotiated using a non-extended BIND, then the compression level is set to RLE compression.

mode_chars.max_decompress_lvl
The maximum decompression level that the Program attempts to negotiate for data flowing supported by the node.

AP_NONE
AP_RLE_COMPRESSION
AP_LZ9_COMPRESSION
AP_LZ10_COMPRESSION
AP_LZ12_COMPRESSION

The level of compression configured cannot be greater than that supported by the node (specified in the field **max_compress_lvl** on START_NODE). Note, if compression is negotiated using a non-extended BIND, then the decompression level is set to LZ9 compression.

mode_chars.comp_in_series
Specifies whether the use of LZ compression preceded by RLE compression is allowed. If this field is set to AP_YES, then **max_compress_lvl** must be set to AP_LZ9_COMPRESSION, AP_LZ10_COMPRESSION, or AP_LZ12_COMPRESSION.

AP_YES

AP_NO

This field cannot be set to AP_YES if the node is configured as not supporting RLE and LZ compression (specified in the field **comp_in_series** on START_NODE).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_COS_NAME

AP_CPSVCMG_ALREADY_DEFD
AP_INVALID_CNOS_SLIM
AP_INVALID_COS_SNASVCMG_MODE
AP_INVALID_DEFAULT_RU_SIZE
AP_INVALID_MAX_NEGOT_SESS_LIM
AP_INVALID_MAX_RU_SIZE_UPPER

AP_INVALID_MAX_RU_SIZE_LOW
AP_RU_SIZE_LOW_UPPER_MISMATCH
AP_INVALID_COMPRESSION
AP_INVALID_MIN_CONWINNERS
AP_INVALID_MIN_CONLOSERS
AP_INVALID_MIN_CONTENTION_SUM
AP_INVALID_MODE_NAME
AP_INVALID_RECV_PACING_WINDOW
AP_INVALID_MAX_RECV_PACING_WIN
AP_INVALID_DEFAULT_RU_SIZES
AP_INVALID_SNASVCMG_MODE_LIMIT
AP_MODE_SESS_LIM_EXCEEDS_NEG
AP_INVALID_CRYPTOGRAPHY
AP_INVALID_MAX_COMPRESS_LVL
AP_INVALID_MAX_DECOMPRESS_LVL
AP_INVALID_COMP_IN_SERIES

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Effects of Redefinition

Following is the effect of redefinition of each field:

description
The updated **description** is returned on subsequent QUERY_MODE verbs.

compression

max_compress_lvl

max_decompress_lvl

comp_in_series

cryptography

max_ru_size_upp

receive_pacing_win

default_ru_size

max_ru_size_low

max_receive_pacing_win
The updated values are used for all subsequent session activation attempts

for this mode and are returned on all subsequent QUERY_MODE verbs. The change does not effect any existing active sessions.

max_neg_sess_lim

plu_mode_session_limit

min_conwin_src

auto_act

min_conloser_src
The updated values are not used for a particular local LU or partner LU pair until the next CNOS command (either locally initiated or remotely initiated). The old value is returned in QUERY_MODE verbs until the next CNOS command.

cos_name
The updated values are used for all subsequent session activation attempts for this mode and are returned on all subsequent QUERY_MODE verbs. The change does not effect any existing active sessions. The updated value is also used for any subsequent mode to COS mapping operation (for example, if this node is a network node and provides mode to COS mapping services or its served end nodes), and is returned on all subsequent QUERY_MODE_TO_COS_MAPPING verbs.

Note: An implicit mode definition can be made explicit by a DEFINE_MODE. This is reflected by subsequent QUERY_MODE verbs returning with **implicit set** to AP_NO.

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. Alternatively, DEFINE_PARTNER_LU can be used to modify all parameters already defined for the partner LU, other than the **fqplu_name** and **plu_alias**.

VCB Structure

```
typedef struct define_partner_lu
{
        unsigned short  opcode;              /* verb operation code          */
        unsigned char   reserv2;             /* reserved                     */
        unsigned char   format;              /* format                       */
        unsigned short  primary_rc;          /* primary return code          */
        unsigned long   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[RD_LEN];
                                             /* resource description         */
        unsigned char   plu_un_name[8];      /* partner LU uninterpreted name */
        unsigned char   preference           /* routing preference           */
        unsigned short  max_mc_ll_send_size; /* max MC send LL size          */
        unsigned char   conv_security_ver;   /* already_verified accepted?   */
        unsigned char   parallel_sess_supp;  /* parallel sessions supported?  */
        unsigned char   reserv2[8];          /* reserved                     */
} PLU_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_PARTNER_LU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

plu_chars.fqplu_name
Fully qualified name of the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

plu_chars.plu_alias
Alias of the partner LU. This is an 8-byte string in a locally displayable character set. This field may be set to all zeros for a partner LU with no alias associated to it.

plu_chars.description
Resource description (returned on QUERY_PARTNER_LU and QUERY_PARTNER_LU_DEFINITION). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

plu_chars.plu_un_name
Uninterpreted name of the partner LU. This is an 8-byte type-A EBCDIC character string.

plu_chars.max_mc_ll_send_size
Maximum size of LL records sent by and received by mapped conversation services at the partner LU. Range: 1–32 767 (32 767 is specified by setting this field to 0)

plu_chars.preference
The preferred routing protocol to be used for session activation to this partner LU. This field can take the following values:

AP_NATIVE
Use native (APPN) routing protocols only.

AP_NONNATIVE
Use nonnative (AnyNet) protocols only.

AP_NATIVE_THEN_NONNATIVE
Try native (APPN) protocols, and if the partner LU cannot be located then retry session activation using nonnative (AnyNet) protocols.

AP_NONNATIVE_THEN_NATIVE
Try nonnative (AnyNet) protocols, and if the partner LU cannot be located then retry session activation using native (APPN) protocols.

AP_USE_DEFAULT_PREFERENCE
Use the default preference defined when the node was started. (This can be recalled by QUERY_NODE.)

Note: Nonnative routing is only meaningful when an AnyNet DLC is available to the Node Operator Facility, and there is an AnyNet link station defined. (See "DEFINE_LS" on page 74.).

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 can set the already verified indicator in an Attach request (AP_YES or AP_NO).

plu_chars.parallel_sess_supp
Specifies whether the partner LU supports parallel sessions (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc

AP_ANYNET_NOT_SUPPORTED
AP_DEF_PLU_INVALID_FQ_NAME
AP_INVALID_UNINT_PLU_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PLU_ALIAS_CANT_BE_CHANGED

AP_PLU_ALIAS_ALREADY_USED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Effects of Redefinition

Following is the effect of redefinition of each field:

fqplu_name
Cannot be changed.

plu_alias
If a previous DEFINE_PARTNER_LU has been issued with a different **plu_alias**, the DEFINE_PARTNER_LU fails. If a previous DEFINE_PARTNER_LU has been issued with an all zero **plu_alias**, the redefinition is accepted and will effect all existing PLU records. If no previous DEFINE_PARTNER_LU has been issued, the specified **plu_alias** is copied into all correspondig implicitly defined partner LU records, unless all zeros are specified, in which case the implicit **plu_aliases** are left uchanged.

Note: Issuing DEFINE_PARTNER_LU with a nonzero **plu_alias** can cause some running applications to fail, if the applicaiton has already obtained the implicit **plu_alias** from an earlier APPC verb and uses it on a subsequent ALLOCATE.

description
The updated **description** is returned on subsequent QUERY_PARTNER_LU verbs.

plu_un_name
The updated **plu_un_name** is used for all subsequent session activation requests to this partner LU, and is returned on all subsequent QUERY_PARTNER_LU verbs.

preference
The updated **preference** is used for all subsequent session activation requests to this partner LU, and is returned on all subsequent QUERY_PARTNER_LU verbs.

max_mc_ll_send_size
The updated **preference** is used for all subsequent session activation requests to this partner LU (even on existing sessions). The change does not effect existing conversations. The updated value is returned on all subsequent QUERY_PARTNER_LU verbs.

conv_security_ver
The updated value is not used for a particular local LU until the number of sessions between that local LU and the partner LU drops to zero. BINDs and RSP(BIND)s will flow using the old setting, and the old value will be returned in QUERY_PARTNER_LU requests until the number of sessions drops to zero. This is because the partner LU can reject subsequent session activation attempts if the security support is different than that of existing active sessions.

parallel_sess_supp
As with **conv_security_ver**, the updated value is not used for a particular local LU until the number of sessions between that local LU and the specified partner LU drops to zero. This is to avoid problems with the architected LU6.2 session consistency check.

Note: An implicit mode definition can be made explicit by a DEFINE_PARTNER_LU. This is reflected by subsequent QUERY_PARTNER_LU verbs returning with **implicit set** to AP_NO.

DEFINE_PORT

DEFINE_PORT defines a new port or modifies an existing one. This port belongs to a specified DLC, which must already have been defined using a DEFINE_DLC verb. The DEFINE_PORT verb provides the port name, which is unique throughout the node, along with port specific parameters and default LS characteristics for use with dynamic link stations. The port specific parameters are concatenated to the basic structure. The default LS characteristics are concatenated immediately following the port specific parameters.

DEFINE_PORT can be used to modify one or more fields on an existing port if the port is in a reset state (after STOP_PORT has been issued) and the **dlc_name** specified on the DEFINE_PORT has not changed since the previous definition of the port.

If the port is active, only the following fields can be modified:

description
implicit_dspu_services
implicit_deact_timer
implicit_cp_cp_sess_support
implicit_link_lvl_error
default_tg_chars
implicit_dspu_template
implicit_ls_limit
link_spec_data_len
link_spec_data

If the port spec data is changed while the port is active, the verb will not be rejected but the modifications will be ignored.

See “DLC Processes, Ports, and Link Stations” on page 14, for more information about the relationship between DLCs, ports, and link stations.

VCB Structure

```
typedef struct define_port
{
        unsigned short  opcode;              /* verb operation code          */
        unsigned char   attributes;          /* verb attributes              */
        unsigned char   format;              /* format                       */
        unsigned short  primary_rc;          /* primary return code          */
        unsigned long   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;         /* resource description         */
        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   implicit_uplink_to_en;/* Implicit links to EN are    */
                                             /* uplink                       */
        unsigned char   reserv3[2];          /* reserved                     */
        unsigned long   port_number;         /* port number                  */
        unsigned short  max_rcv_btu_size;    /* max receive BTU size         */
        unsigned short  tot_link_act_lim;    /* total link activation limit  */
        unsigned short  inb_link_act_lim;    /* inbound link activation limit */
        unsigned short  out_link_act_lim;     /* outbound link activation      */
                                              /* limit                         */
        unsigned char   ls_role;              /* initial link station role     */
        unsigned char   retry_flags;          /* conditions for automatic      */
                                              /* retries                       */
        usigned char    max_activation_attempts;
                                              /* how many automatic retries?   */
        unsigned char   activation_delay_timer;
                                              /* delay between automatic       */
                                              /* retries                       */
        unsigned char   reserv1[10];          /* reserved                       */
        unsigned char   implicit_dspu_template[8];
                                              /* reserved                      */
        unsigned char   implicit_ls_limit;    /* max number of implicit links  */
        unsigned char   reserv2;              /* reserved                      */
        unsigned char   implicit_dspu_services;
                                              /* implicit links support DSPUs  */
        unsigned char   implicit_deact_timer; /* Implicit link HPR link        */
                                              /* deactivation timer            */
        unsigned short  act_xid_exchange_limit;
                                              /* act.  XID exchange limit      */
        unsigned short  nonact_xid_exchange_limit;
                                              /* nonact.  XID exchange limit   */
        unsigned char   ls_xmit_rcv_cap;      /* LS transmit-receive           */
                                              /* capability                    */
        unsigned char   max_ifrm_rcvd;        /* max number of I-frames that   */
                                              /* can be received               */
        unsigned short  target_pacing_count;  /* Target pacing count           */
        unsigned short  max_send_btu_size;    /* Desired max send BTU size     */
        LINK_ADDRESS    dlc_data;             /* DLC data                      */
        LINK_ADDRESS    hpr_dlc_data;         /* HPR DLC data                  */
        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;
                                              /* Implicit links support HPR    */
        unsigned char   implicit_link_lvl_error;
                                              /* Implicit links support HPR    */
                                              /* link-level error recovery     */
        unsigned char   retired1;             /* reserved                      */
        TG_DEFINED_CHARS default_tg_chars;    /* Default TG chars              */
        unsigned char   discovery_supported     /* Discovery function            */
                                              /* supported?                    */
        unsigned short  port_spec_data_len;   /* length of port spec data      */
        unsigned short  link_spec_data_len;   /* length of link spec data      */
} PORT_DEF_DATA;

typedef struct link_address
{
        unsigned short  length;               /* length                        */
        unsigned short  reserve1;             /* reserved                      */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                              /* address                       */
} LINK_ADDRESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_PORT

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

port_name
Name of port being defined. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

port_def_data.description
Resource description (returned on QUERY_PORT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

port_def_data.dlc_name
Name of the associated DLC, which is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This named DLC must have already been defined by a DEFINE_DLC verb.

port_def_data.port_type
Specifies the type of line used by the port. The value corresponds to one of the following line types:

AP_PORT_NONSWITCHED
AP_PORT_SWITCHED
AP_PORT_SATF

Note that if this field is set to AP_PORT_SATF then the **ls_role** must be set to AP_LS_NEG.

port_def_data.port_attributes[0]
This is the bit field. It may take the value AP_NO, or the following:

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 field **port_type** is set to AP_PORT_SWITCHED.

port_def_data.implicit_uplink_to_en
BrNN only: Specifies whether implicit link stations off this port are uplink or downlink if the adjacent node is an end node. The value of this field will only be considered if there are no existing links to the same partner, as such links are used first to determine the link type.

AP_NO
Implicit links are downlink.

AP_YES
Implicit links are uplink.

Other node types: This field is ignored.

port_def_data.port_number
Port number.

port_def_data.max_rcv_btu_size
Maximum BTU size that can be received. If implicit HPR-capable links are not supported on the port, then this must be set to a value greater than or equal to 99. If implicit HPR-capable links are supported on the port, then this must be set to a value greater than or equal to 768.

port_def_data.tot_link_act_lim
Total link activation limit. This specifies the maximum number of link stations that can be active concurrently. This must be greater than or equal to the sum of the **inb_link_act_lim** and **out_link_act_lim** fields. If the **port_type** is set to AP_PORT_NONSWITCHED and the **ls_role** is set to AP_LS_NEG or AP_LS_SEC then this field must be set to one. If the **ls_role** is set to AP_LS_PRI then this field must be in the range greater than or equal to one to 256. If this port is for the AnyNet DLC, you must use 65535.

port_def_data.inb_link_act_lim
Inbound link activation limit. This specifies the number of link stations reserved for inbound activation on this port. The maximum number of outbound link stations that can be active concurrently is therefore **port_def_data.tot_link_act_lim** - **port_def_data.inb_link_act_lim**. If the **port_type** is set to AP_PORT_NONSWITCHED and the **ls_role** is set to AP_LS_NEG or AP_LS_PRI then this field must be set to zero. If the **port_type** is set to AP_PORT_NONSWITCHED and the **ls_role** is set to AP_LS_SEC then this field must be set to zero or one. If this port is for the AnyNet DLC, you must use zero.

port_def_data.out_link_act_lim
Outbound link activation limit. This specifies the number of link stations reserved for outbound activation on this port. The maximum number of inbound link stations that can be active concurrently is therefore **port_def_data.tot_link_act_lim** - **port_def_data.out_link_act_lim**. If the **port_type** is set to AP_PORT_NONSWITCHED and the **ls_role** is set to AP_LS_NEG then this field must be set to zero. If the **ls_role** is set to AP_LS_PRI then this field must be equal to **tot_link_act_lim**. If the **port_type** is set to AP_PORT_NONSWITCHED and the **ls_role** is set to AP_LS_SEC then this field must be set to zero or one. If this port is for the AnyNet DLC, you must use zero.

port_def_data.ls_role
Link station role. This can be negotiable (AP_LS_NEG), primary (AP_LS_PRI), or secondary (AP_LS_SEC). The link station role determines the relationship between the values specified by the **tot_act_lim**, **inb_link_act_lim**, and **out_link_act_lim** fields as described above. Note that if the **port_type** is set to AP_PORT_SATF then the **ls_role** must be set to AP_LS_NEG.

port_def_data.retry_flags
This field specifies the conditions under which activation of this link station is subject to automatic retry if the flag AP_INHERIT_RETRY is set on DEFINE_LS in **port_def_data.retry_flags**. It is a bit field, and may take any of the following values bit-wise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, APPN will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, APPN attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_INHERIT_RETRY
In addition to the retry conditions specified by flags in this field, those specified in the **retry_flags** field of the underlying port definition will also be used.

port_def_data.max_activation_attempts
This field has no effect unless at least one flag is set in DEFINE_LS in **port_def_data.retry_flags** and **port_def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means no limit. The value AP_USE_DEFAULTS results in the use of **max_activiation_attempts** supplied on DEFINE_DLC.

port_def_data.activation_delay_timer
This field has no effect unless at least one flag is set in DEFINE_LS in **port_def_data.retry_flags** and **activiation_delay_timer** on DEFINE_LS is set to AP_USE_DEFAULTS.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **port_def_data.retry_flags**.

The value AP_USE_DEFAULTS results in the use of **activiation_delay_timer** supplied on DEFINE_DLC.

If zero is specified, the Program uses a default timer duration of thirty seconds.

port_def_data.implicit_dspu_template
Specifies the DSPU template, defined with the DEFINE_DSPU_TEMPLATE verb, that 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 fails. This is an 8-byte string in a locally-displayable character set. All 8 bytes are significant and must be set.

If the **port_def_data.implicit_dspu_services** field is not set to AP_PU_CONCENTRATION, then this field is reserved.

port_def_data.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. A value of 0 means that there is no limit, a value of AP_NO_IMPLICIT_LINKS means that no implicit links are allowed.

port_def_data.implicit.dspu_services
Specifies the services that the local node will provide to the downstream PU across implicit links activated on this port. This is set to one of the following values:

AP_DLUR
Local node will provide DLUR services for the downstream PU (using the default DLUS configured through the DEFINE_DLUR_DEFAULTS verb). This setting is only valid if the local node is a network node.

AP_PU_CONCENTRATION
Local node will provide PU Concentration for the downstream PU (and will put in place definitions as specified by the DSPU template specified in the field **port_def_data.implicit_dspu_template**).

AP_NONE
Local node will provide no services for this downstream PU.

port_def_data.implicit_deact_timer
Limited resource link deactivation timer (in seconds). If **implicit_limited_resource** is set to AP_YES or AP_NO_SESSIONS, then an HPR-capable implicit link is automatically deactivated if no data traverses the link for the duration of this timer, and no sessions are using the link.

If **implicit_limited_resource** is set to AP_INACTIVITY then an implicit link is automatically deactivated if no data traverses the link for the duration of this timer.

The value is an integer in the range of 0–1000 seconds. The default is 10 seconds.

If zero is specified, the default value of 30 is used. Otherwise the minimum value is 5. (If it is set any lower, the specified value will be ignored and 5 will be used.) Note that this parameter is reserved unless **implicit_limited_resource** is set to AP_NO.

port_def_data.act_xid_exchange_limit
Activation XID exchange limit.

port_def_data.nonact_xid_exchange_limit
Non-activation XID exchange limit.

port_def_data.ls_xmit_rcv_cap
Specifies the link station transmit/receive capability. This is either two-way simultaneous (AP_LS_TWS) (also known as duplex or full-duplex) or two way alternating (AP_LS_TWA) (also know as half-duplex).

port_def_data.max_ifrm_rcvd
Maximum number of I-frames that can be received by the local link stations before an acknowledgment is sent. The range is 1–127.

port_def_data.target_pacing_count
Numeric value between 1 and 32 767 inclusive indicating the desired pacing window size for BINDs on this TG. The number is only significant

when fixed bind pacing is being performed. Note that Personal Communications or Communications Server does not currently use this value.

port_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 than can be transmitted between a link station pair. If implicit HPR-capable links are not supported on the port then this must be set to a value greater than or equal to 99. If implicit HPR-capable links are supported on the port then this must be set to a value greater than or equal to 768.

port_def_data.dlc_data.length
Port address length.

port_def_data.dlc_data.address
Port address.

port_def_data.hpr_dlc_data.length
HPR Port address length.

port_def_data.hpr_dlc_data.address
HPR Port address. This is currently used when supporting HPR links. The field specifies the information sent by Personal Communications or Communications Server in the X'80' subfield of the X'61' control vector on XID3s exchanged on link stations using this port. It is passed on the ACTIVATE_PORT issued to the DLC by Personal Communications or Communications Server. Some DLCs can require this information to be filled in for ports supporting HPR links.

port_def_data.implicit_cp_cp_sess_support
Specifies whether CP-CP sessions are permitted for implicit link stations off this port (AP_YES or AP_NO).

port_def_data.implicit_limited_resource
Specifies whether implicit link stations off this port should be deactivated when there are no sessions using the link. This is set to one of the following values:

AP_NO
Implicit links are not limited resources and will not be deactivated automatically.

AP_YES or AP_NO_SESSIONS
Implicit links are a limited resource and will be deactivated automatically when no active sessions are using them.

AP_INACTIVITY
Implicit links are a limited resource and will be deactivated automatically when no active sessions are using them, or when no data has followed on the link for the time period specified by the **implicit_deact_timer** field.

port_def_data.implicit_hpr_support
Specifies whether HPR should be supported on implicit links (AP_YES or AP_NO).

port_def_data.implicit_link_lvl_error
Specifies whether HPR traffic should be sent on implicit links using link-level error recovery (AP_YES or AP_NO). Note that the parameter is reserved if **implicit_hpr_support** is set to AP_NO.

port_def_data.default_tg_chars
TG characteristics (See "DEFINE_COS" on page 35). These are used for implicit link stations off this port and also for defined link stations that specify **use_default_tg_chars**.

port_def_data.discovery_supported
Specifies whether Discovery functions are to be performed on this port (AP_YES or AP_NO).

port_def_data.port_spec_data_len
Length of data to be passed unchanged to port on ACTIVATE_PORT signal. The data should be concatenated to the basic structure.

port_def_data.link_spec_data_len
This field should always be set to zero.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PORT_NAME

AP_INVALID_DLC_NAME
AP_INVALID_PORT_TYPE
AP_INVALID_BTU_SIZE
AP_INVALID_LS_ROLE
AP_INVALID_LINK_ACTIVE_LIMIT
AP_INVALID_MAX_IFRM_RCVD
AP_INVALID_DSPU_SERVICES
AP_HPR_NOT_SUPPORTED
AP_DLUR_NOT_SUPPORTED
AP_PU_CONC_NOT_SUPPORTED
AP_INVALID_TEMPLATE_NAME
AP_INVALID_RETRY_FLAGS
AP_INVALID_IMPLICIT_UPLINK

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PORT_ACTIVE

AP_DUPLICATE_PORT_NUMBER
AP_CANT_MODIFY_WHEN_ACTIVE
AP_CANT_MODIFY_VISIBILITY
AP_INVALID_IMPLICIT_UPLINK

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_TP

The DEFINE_TP verb defines transaction program (TP) information for use by the Node Operator Facility TP Attach Manager when it processes incoming attaches from partner LUs. This verb can also be used to modify one or more fields on a previously defined transaction program (but cannot be used to modify Personal Communications or Communications Server defined transaction programs).

VCB Structure

```
typedef struct define_tp
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   attributes;       /* verb attributes              */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   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[RD_LEN]
                                          /* resource description         */
        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 supported             */
        unsigned char   reserv3[9];       /* reserved                     */
        unsigned short  tp_instance_limit; /* limit on currently active TP  */
                                          /* instances                    */
        unsigned short  incoming_alloc_timeout;
                                          /* incoming allocation timeout  */
        unsigned short  rcv_alloc_timeout; /* receive allocation timeout   */
        unsigned short  tp_data_len;      /* TP data length               */
        TP_SPEC_DATA    tp_data;          /* TP data                      */
} TP_CHARS;
typedef struct tp_spec_data
{
        unsigned char   pathname[256];    /* path and TP name             */
        unsigned char   parameters[64];   /* parameters for TP            */
        unsigned char   queued;           /* queued TP                    */
        unsigned char   load_type;        /* type of load-DETACHED/CONSOLE */
        unsigned char   dynamic_load      /* dynamic loading of TP enabled */
        unsigned char   reserved[5];      /* reserved                     */
} TP_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_TP

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

tp_name
: Name of the transaction program (TP) being defined. This is a 64-byte EBCDIC string padded to the right with EBCDIC spaces. Note that Personal Communications or Communications Server does not check the character set of this field.

tp_chars.description
: Resource description (returned on QUERY_TP_DEFINITION and QUERY_TP). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

tp_chars.conv_type
: Specifies the types of conversation supported by this transaction program.

AP_BASIC
AP_MAPPED
AP_EITHER

tp_chars.security_rqd
: Specifies whether conversation security information is required to start the transaction program (AP_NO or AP_YES).

tp_chars.sync_level
: Specifies the synchronization levels supported by the transaction program.

AP_NONE
: The transaction program supports a synchronization level of None.

AP_CONFIRM_SYNC_LEVEL
: The transaction program supports a synchronization level of Confirm.

AP_EITHER
: The transaction program supports a synchronization level of None or Confirm.

AP_SYNCPT_REQUIRED
: The transaction program supports a synchronization level of Sync-point.

AP_SYNCPT_NEGOTIABLE
: The transaction program supports a synchronization level of None, Confirm or Sync-point.

tp_chars.dynamic_load
: Specifies whether the transaction program can be dynamically loaded (AP_YES or AP_NO).

tp_chars.enabled
: Specifies whether the transaction program can be attached successfully (AP_YES or AP_NO). The default is AP_NO.

tp_chars.pip_allowed
: Specifies whether the transaction program can receive program initialization (PIP) parameters (AP_YES or AP_NO).

tp_chars.duplex_support
Indicates whether the transaction program is full or half duplex.

AP_FULL_DUPLEX
Specifies that the transaction program is full duplex.

AP_HALF_DUPLEX
Specifies that the transaction program is half duplex.

AP_EITHER_DUPLEX
Specifies that the transaction program can be either half or full duplex

tp_chars.tp_instance_limit
Limit on the number of concurrently active transaction program instances. A value of zero means no limit.

tp_chars.incoming_alloc_timeout
Specifies the number of seconds that an incoming attach will be queued waiting for a RECEIVE_ALLOCATE. Zero implies no timeout, and so it will be held indefinitely.

tp_chars.rcv_alloc_timeout
Specifies the number of seconds that a RECEIVE_ALLOCATE verb will be queued while waiting for an Attach. Zero implies no timeout, and so it will be held indefinitely.

tp_chars.tp_data_len
Length of the implementation-dependent transaction program data.

tp_spec_data
Information used by the Attach Manager when launching the transaction program. Refer to the information about Attach Manager in *Personal Communications for Windows, Version 5.7 Client/Server Communications Programming* for further details of how this is used.

tp_spec_data.pathname
Specifies the path and transaction program name.

tp_spec_data.parameters
Specifies the parameters for the transaction program.

tp_spec_data.queued
Specifies whether the transaction program will be queued.

tp_spec_data.load_type
Specifies whether type of load is either AP_AM_CONSOLE, AP_AM_DETACHED or AP_AM_WINDOW.

tp_spec_data.dynamic_load
Specifies how the transaction program will be loaded.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_SYSTEM_TP_CANT_BE_CHANGED

AP_INVALID_CONV_TYPE
AP_INVALID_SYNC_LEVEL
AP_INVALID_DYNAMIC_LOAD
AP_INVALID_ENABLED
AP_INVALID_PIP_ALLOWED
AP_INVALID_DUPLEX_SUPPORT

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_CANT_MODIFY_VISIBILITY

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Effects Of Redefinition: The redefinition of each field takes effect immediately (for example, when the next instance of the transaction program is started). However, changes to the fields **incoming_alloc_timeout** and **rcv_alloc_timeout** will not effect any attaches or RECEIVE_ALLOCATES that are already queued.

DELETE_ADJACENT_NODE

DELETE_ADJACENT_NODE removes entries in the node directory database that are associated with the resources on an adjacent node.

To remove the node's control point from the directory along with its LUs, set **num_of_lus** to zero. If **num_of_lus** is nonzero, this verb is used to remove node LUs from the directory, leaving the control point definition intact.

If the verb fails for any reason, no directory entries will be deleted.

VCB Structure

The DELETE_ADJACENT_NODE verb contains a variable number of ADJACENT_NODE_LU overlays. The ADJACENT_NODE_LU structures are concatenated onto the end of DELETE_ADJACENT_NODE structure.

```
typedef struct delete_adjacent_node
{
        unsigned short  opcode;          /* verb operation code       */
        unsigned char   reserv2;         /* reserved                  */
        unsigned char   format;          /* format                    */
        unsigned short  primary_rc;      /* primary return code       */
        unsigned long   secondary_rc;    /* secondary return code     */
        unsigned char   cp_name[17];     /* CP name                   */
        unsigned short  num_of_lus;      /* number of LUs             */
} DELETE_ADJACENT_NODE;
```

```
typedef struct adjacent_node_lu
{
        unsigned char   wildcard_lu;     /* wildcard LU name indicator */
        unsigned char   fqlu_name[17];   /* fully qualified LU name    */
        unsigned char   reserv1[6];      /* reserved                   */
} ADJACENT_NODE_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DELETE_ADJACENT_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

cp_name
The fully qualified name of the control point in the adjacent LEN end node. The name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

num_of_lus
The number of LUs to be deleted. Set this to zero if the entire node definition is to be deleted. This number represents the number of adjacent LU overlays that follow the DELETE_ADJACENT_NODE VCB.

adjacent_node_lu.wildcard_lu
Indicates whether the specified LU name is a wildcard name (AP_YES or AP_NO).

adjacent_node_lu.fqlu_name
The LU name to be deleted. If this name is not fully qualified, the network ID of the CP name is assumed. The name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of one or two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CP_NAME

AP_INVALID_LU_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_CP_NAME

AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_CN

DELETE_CN deletes and frees the memory for a connection network control block if all the associated ports are reset. DELETE_CN can also be used to delete selected ports from a connection network. To do this, the user should set the **num_ports** field to a nonzero value and supply the port names of the ports to be deleted.

VCB Structure

```
typedef struct delete_cn
{
        unsigned short  opcode;         /* verb operation code       */
        unsigned char   attributes;     /* verb attributes           */
        unsigned char   format;         /* format                    */
        unsigned short  primary_rc;     /* primary return code       */
        unsigned long   secondary_rc;   /* secondary return code     */
        unsigned char   fqcn_name[17];  /* name of connection network */
        unsigned char   reserv1;        /* reserved                  */
        unsigned short  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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

fqcn_name
Name of connection network (17 bytes long) to be deleted. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

num_ports
The number of ports to delete on the connection network. This should be set to zero if the entire connection network is to be deleted.

port_name
Names of the ports to be deleted if the **num_ports** is nonzero. Each port name is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If the **num_ports** field is zero this field is reserved.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CN_NAME

AP_INVALID_NUM_PORTS_SPECIFIED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_COS

DELETE_COS deletes a class-of-service entry unless it is one of the default classes of service defined by SNA.

VCB Structure

```
typedef struct delete_cos
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

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

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_COS_NAME_NOT_DEFD

AP_SNA_DEFD_COS_CANT_BE_DELETE

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_DLC

DELETE_DLC deletes all ports, link stations, and connection network transmission groups (TGs) associated with the DLC if it is reset. All DLC control blocks are deleted and the memory freed. The Node Operator Facility returns a response specifying whether the DLC was deleted successfully.

Note that if a link station, which has a PU associated with it, is deleted (because it is associated with the DLC) then any LUs defined on this PU will also be deleted.

VCB Structure

```
typedef struct delete_dlc
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dlc_name
Name of DLC to be deleted. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLC_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_DLC_ACTIVE

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_DOWNSTREAM_LU

This verb applies only to Communications Server.

VCB Structure

```
typedef struct delete_downstream_lu
{
        unsigned short  opcode;           /* verb operation code       */
        unsigned char   attributes;       /* verb attributes           */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* primary return code       */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

The other values that can be bit-wise ORed into this field are as follows:

AP_DELAY_IF_REQUIRED
This specifies that the downstream LU specified by **dslu_name** is currently active, this verb should be queued inside the Program until the LU becomes inactive. In this case, the verb is processed to completion when the LU becomes inactive.

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dslu_name
Name of the downstream LU that is being deleted. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_DSLU_ACTIVE
AP_DELAYED_VERB_PENDING

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_DOWNSTREAM_LU_RANGE

This verb applies only to Communications Server.

For example, a base name of LUNME combined with an NAU range of 1 to 4 deletes the LUs LUNME001, LUNME002, LUNME003, and LUNME004. A base name of less than five non-pad characters results in LU names of less than eight non-pad characters.

This verb deletes all LUs in the range. If an LU in the range does not exist, then the verb continues with the next one that does exist. The verb only fails if no LUs exist in the specified range.

VCB Structure

```
typedef struct delete_downstream_lu_range
{
        unsigned short  opcode;           /* verb operation code       */
        unsigned char   attributes;       /* verb attributes           */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* primary return code       */
        unsigned long   secondary_rc;     /* secondary return code     */
        unsigned char   dslu_base_name[5];/* Downstream LU base name   */
        unsigned char   min_nau;          /* min NAU address in range  */
        unsigned char   max_nau;          /* max NAU address in range  */
} DELETE_DOWNSTREAM_LU_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DELETE_DOWNSTREAM_LU_RANGE

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dslu_base_name
Base name for downstream LU name range. This is a 5-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three type-A EBCDIC numeric characters, representing the decimal value of the NAU address, for each LU in the NAU range.

min_nau
Minimum NAU address in the range. This can be from 1 to 255 inclusive.

max_nau
Maximum NAU address in the range. This can be from 1 to 255 inclusive.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_NAU_ADDRESS

AP_INVALID_LU_NAME

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

primary_rc
AP_STATE_CHECK

AP_INVALID_LU_NAME
AP_DSLU_ACTIVE
AP_DELAYED_VERB_PENDING

secondary_rc
AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_DSPU_TEMPLATE

This verb applies only to Communications Server.

VCB Structure

Format 1

```
typedef struct delete_dspu_template
{
        unsigned short  opcode;              /* verb operation code       */
        unsigned char   attributes;          /* verb attributes           */
        unsigned char   format;              /* format                    */
        unsigned short  primary_rc;          /* primary return code       */
        unsigned long   secondary_rc;        /* secondary return code     */
        unsigned char   template_name[8];    /* name of template          */
        unsigned short  num_of_dslu_templates;
                                             /* Number of DSLU templates  */
        unsigned char   reserv1[10];         /* reserved                  */
} DELETE_DSPU_TEMPLATE;
```

```
typedef struct dslu_template
{
        unsigned char   min_nau;             /* min NAU address in range  */
        unsigned char   max_nau;             /* max 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;
```

VCB Structure

Format 0

```
typedef struct delete_dspu_template
{
        unsigned short  opcode;              /* verb operation code       */
        unsigned char   attributes;          /* verb attributes           */
        unsigned char   format;              /* format                    */
        unsigned short  primary_rc;          /* primary return code       */
        unsigned long   secondary_rc;        /* secondary return code     */
        unsigned char   template_name[8];    /* name of template          */
} DELETE_DSPU_TEMPLATE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DELETE_DSPU_TEMPLATE

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibilityof the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

template_name
Name of the DSPU template. (This corresponds to the name specified in the **implicit_dspu_template** field on PORT_DEF_DATA). This is an 8–byte string in a locally-displayable character set. All 8 bytes are significant and must be set.

num_of_dslu_templates
The number of DSLU template overlays which follow the DEFINE_DSPU_TEMPLATE VCB. This can be from 0 to 255 inclusive. The DSLU templates are appended as overlays to the end of the DELETE_DSPU_TEMPLATE VCB.

dslu_template.min_nau
Minimum NAU address in the range. This can be from 1 to 255 inclusive.

dslu_template.max_nau
Maximum NAU address in the range. This can be from 1 to 255 inclusive.

def_data.allow_timeout
This field is reserved.

def_data.delayed_logon
This field is reserved.

dslu_template.host_lu
This field is reserved.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TEMPLATE_NAME

AP_INVALID_NAU_RANGE

If the verb does not execute because one or more relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_FOCAL_POINT

The DELETE_FOCAL_POINT verb can be used to delete focal points of a specified type and category. For more information about focal point types, see “DEFINE_FOCAL_POINT” on page 61. If an active focal point is deleted it will be revoked. To revoke the active focal point (of any type) specify a type of AP_ACTIVE. If a backup or implicit focal point is deleted (by specifying AP_BACKUP or AP_IMPLICIT) when it is not currently active, any information stored about it will simply be removed.

Note that the DEFINE_FOCAL_POINT verb can also be used to revoke currently active focal points. This duplicated function is retained for back compatibility.

VCB Structure

```
typedef struct delete_focal_point
{
        unsigned short  opcode;         /* verb operation code         */
        unsigned char   reserv2;        /* reserved                    */
        unsigned char   format;         /* format                      */
        unsigned short  primary_rc;     /* primary return code         */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

ms_category
Management services category. This cab either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services categories as described in SNA management services, or an 8-byte type 1134 EBCDIC installation-defined name.

type Specifies the type of the focal point that is being deleted. Possible types are:

AP_ACTIVE
The currently active focal point (which can be of any type) is revoked.

AP_IMPLICIT
The implicit definition is removed. If the currently active focal point is an implicit focal point, then it is revoked.

AP_BACKUP
The backup definition is removed. If the currently active focal point is a backup focal point, then it is revoked.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TYPE

AP_INVALID_CATEGORY_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_INTERNAL_PU

The DELETE_INTERNAL_PU verb requests the deletion of a DLUR-served local PU. The verb will only succeed if the PU does not have an active SSCP-PU session.

Any LUs associated with the PU will be deleted.

VCB Structure

```
typedef struct delete_internal_pu
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
Name of the internal PU that is being deleted. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_TYPE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PU_NOT_RESET

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LOCAL_LU

The DELETE_LOCAL_LU verb requests deletion of the local LU definition.

VCB Structure

```
typedef struct delete_local_lu
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
Name of the local LU that is being defined. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_CANT_DELETE_CP_LU

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LS

DELETE_LS checks that the link station has been previously defined and reset. It removes the link station control block and returns a response from the Node Operator Facility specifying whether the link station has been deleted successfully. Note that any LUs defined on the PU using this link station will also be deleted.

VCB Structure

```
typedef struct delete_ls
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

ls_name
Name of link station being deleted. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LINK_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LS_ACTIVE

AP_INVALID_LINK_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LU_0_TO_3

This verb is used to delete a specific LU.

VCB Structure

```
typedef struct delete_lu_0_to_3
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
Name of the LU to be deleted. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_CANT_DELETE_IMPLICIT_LU

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LU_0_TO_3_RANGE

This verb is used to delete a range of LUs. The node operator provides a base name and an NAU range. The LU names are generated by combining the base name with the NAU addresses.

For example, a base name of LUNME combined with an NAU range of 1 to 4 would delete the LUs LUNME001, LUNME002, LUNME003, and LUNME004. A base name of less than five non-pad characters results in LU names of less than eight non-pad characters.

All LUs in the range are deleted. If an LU in the range does not exist, then the verb continues with the next one that does exist. The verb fails if no LUs exist in the specified range.

VCB Structure

Format 1

```
typedef struct delete_lu_0_to_3_range
{
        unsigned short  opcode;                /* verb operation code       */
        unsigned char   attributes;            /* verb attributes           */
        unsigned char   format;                /* format                    */
        unsigned short  primary_rc;            /* primary return code       */
        unsigned long   secondary_rc;          /* secondary return code     */
        unsigned char   base_name[6];          /* base name                 */
        unsigned char   min_nau;               /* minimum NAU address       */
        unsigned char   max_nau;               /* maximum NAU address       */
        unsigned char   name_attributes;       /* Attributes of base_name   */
        unsigned char   base_number;           /* Base number for LU names  */
        unsigned char   reserv5[16];               /* reserved                  */
} DELETE_LU_0_TO_3_RANGE;
```

VCB Structure

Format 0

```
typedef struct delete_lu_0_to_3_range
{
        unsigned short  opcode;                /* verb operation code       */
        unsigned char   attributes;            /* verb attributes           */
        unsigned char   format;                /* format                    */
        unsigned short  primary_rc;            /* primary return code       */
        unsigned long   secondary_rc;          /* secondary return code     */
        unsigned char   base_name[5];          /* base name                 */
        unsigned char   min_nau;               /* minimum NAU address       */
        unsigned char   max_nau;               /* maximum NAU address       */
        unsigned char   reserv3;               /* reserved                  */
} DELETE_LU_0_TO_3_RANGE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DELETE_LU_0_TO_3_RANGE

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

base_name
: Base LU name. This is an 5-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three type-A EBCDIC numeric characters, representing the decimal value of the NAU address, for each LU in the NAU range.

min_nau
: Minimum NAU address in the range. This can be from 1 to 255 inclusive.

max_nau
: Maximum NAU address in the range. This can be from 1 to 255 inclusive.

name_attributes
: This bit field modifies the interpretation and usage of the supplied **base_name**. This field may take the value of zero, or any or all of the following values bit-wise ORed together.

  AP_USE_HEX_IN_NAME
  : If this bit is set, the interpretation of the **base_name** is modified as follows:

    This is a 6–byte alphanumeric types A EBCDID string (starting with a letter), padded to the right with EBCDID spaces. This base name is appended with two EBCDID characters, representing the hexadecimal values of the NAU address, for each LU in the NAU range.

  AP_USE_BASE_NUMBER
  : If this bit is set, the interpretation of the **base_name** is modified as follows:

    This is a 5–byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This base name is appended with three EBCDIC numeric characters, representing the decimal index of the LU in the range, starting with **base_number** and ending with (**base_number** + **max_nau** + **min_nau**).

base_number
: If the AP_USE_BASE_NUMBER bit is not set in **name_attributes** this field is ignored. Otherwise, this field modifies the interpretation of **base_name** as described above. Legal values are from zero to (255 **– max_nau** + **min_nau**).

Returned Parameters

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

primary_rc
: AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_NAU_ADDRESS

AP_INVALID_LU_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_CANT_DELETE_IMPLICIT_LU

If the verb does not execute because the system has not been built with dependent LU support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LU_POOL

This verb is used to delete an LU pool or to remove LUs from a pool. If no LU names are specified, the entire pool is removed. This verb completes successfully when the specified LUs within the LU pool, or the LU pool itself, no longer exist. The verb only fails if none of the specified LUs exist, or if there are no LUs in the specified pool.

VCB Structure

```
typedef struct delete_lu_pool
{
        unsigned short  opcode;             /* verb operation code       */
        unsigned char   attributes;         /* verb attributes           */
        unsigned char   format;             /* format                    */
        unsigned short  primary_rc;         /* primary return code       */
        unsigned long   secondary_rc;       /* secondary return code     */
        unsigned char   pool_name[8];       /* LU pool name              */
        unsigned short  num_lus;            /* number of LUs to add      */
        unsigned char   lu_names[10][8];    /* LU names                  */
} DELETE_LU_POOL;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DELETE_LU_POOL

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pool_name
Name of the LU pool. All 8 bytes are significant and must be set. This name is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

num_lus
Number of LUs specified in the **lu_names** list.

lu_names
Names of the LUs to be removed. Each name is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_POOL_NAME

AP_INVALID_LU_NAME
AP_INVALID_NUM_LUS

If the verb does not execute because the system has not been built with dependent LU support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_MODE

The DELETE_MODE verb requests deletion of a mode definition. Default definitions for CPSVCMG, SNASVCMG, and other standard SNA modes will not be deleted.

VCB Structure

```
typedef struct delete_mode
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

mode_name
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.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_CP_OR_SNA_SVCMG_UNDELETABLE

AP_MODE_UNDELETABLE
AP_DEL_MODE_DEFAULT_SPCD
AP_MODE_NAME_NOT_DEFD

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_PARTNER_LU

The DELETE_PARTNER_LU requests the deletion of a partner LU definition.

VCB Structure

```
typedef struct delete_partner_lu
{
        unsigned short  opcode;         /* verb operation code     */
        unsigned char   reserv2;        /* reserved                */
        unsigned char   format;         /* format                  */
        unsigned short  primary_rc;     /* primary return code     */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

fqplu_name
Fully qualified name of the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PLU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_PORT

DELETE_PORT deletes all link stations and connection network transmission groups (TGs) associated with the port if it is reset. It then deletes the port's control block, frees the memory, and returns a response from the Node Operator Facility indicating whether the port has been deleted successfully.

Note that if a link station, which has a PU associated with it, is deleted (because it is associated with the port) then any LUs defined on this PU will also be deleted.

VCB Structure

```
typedef struct delete_port
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

port_name
Name of port being deleted. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PORT_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PORT_ACTIVE

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_TP

The DELETE_TP requests the deletion of a transaction program (TP) definition.

VCB Structure

```
typedef struct delete_tp
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   attributes;     /* verb attributes       */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

tp_name
Name of the transaction program. The Program does not check the character set of this field.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TP_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 5. Activation and Deactivation Verbs

This chapter describes verbs that are used to activate and deactivate:

- Data link controls (DLCs)
- Internal PUs
- Ports
- Link stations
- Sessions
- Conversation groups

This chapter also describes a verb used to request a path switch to a connection that supports High-Performance Routing (HPR).

START_DLC

START_DLC requests the activation of a data link control (DLC). It is subsequently returned indicating whether the activation of the DLC was successful. Note that the DLC can be started even if no ports have been defined for it. See "DLC Processes, Ports, and Link Stations" on page 14, for more information about the relationship between DLCs, ports, and link stations.

VCB Structure

```
typedef struct start_dlc
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

dlc_name
Name of Data Link Control instance that is to be started. This is an 8-byte string in a locally displayable character set, which must have already been defined by a DEFINE_DLC verb.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLC

If the verb does not execute because the DLC is deactivating, the Program returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
AP_DLC_DEACTIVATING

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

START_INTERNAL_PU

The START_INTERNAL_PU verb requests the dependent LU requester (DLUR) to initiate SSCP-PU session activation for a previously defined local PU that is served by DLUR.

VCB Structure

```
typedef struct start_internal_pu
{
        unsigned short  opcode;             /* verb operation code   */
        unsigned char   reserv2;            /* reserved              */
        unsigned char   format;             /* format                */
        unsigned short  primary_rc;         /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
Name of the internal PU for which the SSCP-PU session activation flows will be solicited. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlus_name
Name of the dependent LU server (DLUS) node that DLUR will contact to solicit SSCP-PU session activation for the given PU. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This value overrides the value specified in the DEFINE_INTERNAL_PU verb. If the field is set to all zeros, the DLUS specified in the DEFINE_INTERNAL_PU verb will be used. If no DLUS has been specified in the DEFINE_INTERNAL_PU verb, then the global default (if specified by a DEFINE_DLUR_DEFAULTS verb) will be used.

bkup_dlus_name
Name of the DLUS node that DLUR will store as the backup DLUS for the given PU. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This value overrides the value specified in the DEFINE_INTERNAL_PU verb. If the field is set to all zeros, the backup DLUS name specified by a DEFINE_INTERNAL_PU verb will be retained as the backup DLUS for this PU. If no backup DLUS was specified by the DEFINE_INTERNAL_PU verb, the global backup default DLUS (if defined by the DEFINE_DLUR_DEFAULTS verb) is retained as the backup default for this PU.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLUS_NAME

AP_INVALID_BKUP_DLUS_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_NO_DEFAULT_DLUS_DEFINED

AP_PU_NOT_DEFINED
AP_PU_ALREADY_ACTIVATING
AP_PU_ALREADY_ACTIVE

If the verb does not execute successfully, the Program returns the following parameters:

primary_rc
AP_UNSUCCESSFUL

secondary_rc
AP_DLUS_REJECTED

AP_DLUS_CAPS_MISMATCH
AP_PU_FAILED_ACTPU

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

START_LS

START_LS requests activation of a link. It is returned as a response specifying whether the link was successfully activated.

See "DLC Processes, Ports, and Link Stations" on page 14, for more information about the relationship between DLCs, ports and link stations.

VCB Structure

```
typedef struct start_ls
{
        unsigned short  opcode;         /* verb operation code       */
        unsigned char   reserv2;        /* reserved                  */
        unsigned char   format;         /* format                    */
        unsigned short  primary_rc;     /* primary return code       */
        unsigned long   secondary_rc;   /* secondary return code     */
        unsigned char   ls_name[8];     /* name of link station      */
        unsigned char   enable;         /* whether the link is enabled*/
        unsigned char   reserv3[3];     /* reserved                  */
} START_LS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_START_LS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

ls_name
Name of link station to be started. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. The value of **ls_name** must match that on the DEFINE_LS verb.

enable
Set this field to start the link. If this field is set to AP_ACTIVATE, then the link is started. Otherwise, the link is not started, and the following values are possible. These values can be ORed together.

AP_AUTO_ACT
The link can subsequently be activated on demand by the local node. This value is only valid if **auto_act_supp** was set to AP_YES on the DEFINE_LS verb.

AP_REMOTE_ACT
The link can subsequently be activated by the remote node. This does not alter the defined value of **disable_remote_act**.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LINK_NAME_SPECIFIED

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PORT_INACTIVE

AP_ACTIVATION_LIMITS_REACHED
AP_PARALLEL_TGS_NOT_SUPPORTED
AP_ALREADY_STARTING
AP_LINK_DEACT_IN_PROGRESS

If the verb does not execute because it was canceled by a subsequent STOP_LS or STOP_PORT before the link became active, the Program returns the following parameters:

primary_rc
AP_CANCELLED

secondary_rc
AP_LINK_DEACTIVATED

If the verb does not execute because the partner could not be found by the link software, the Program returns the following parameters:

primary_rc
AP_LS_FAILURE

secondary_rc
AP_PARTNER_NOT_FOUND

If the verb does not execute because a link error occurred while the link was being established, the Program returns the following parameters:

primary_rc
AP_LS_FAILURE

secondary_rc
AP_ERROR

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

START_PORT

START_PORT requests the activation of a port. It is returned indicating whether the port was successfully activated. The port can be started even if no link stations have been defined for it, but it will not be started if its parent DLC is inactive.

See “DLC Processes, Ports, and Link Stations” on page 14, for more information about the relationship between DLCs, ports and link stations.

VCB Structure

```
typedef struct start_port
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

port_name
Name of port to be started. This is an 8-byte string in a locally displayable character set and must match that on the DEFINE_PORT verb.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PORT_NAME

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_DLC_INACTIVE

AP_STOP_PORT_PENDING
AP_DUPLICATE_PORT

If the verb does not execute because it was canceled, the Program returns the following parameter:

primary_rc
AP_CANCELLED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

STOP_DLC

STOP_DLC requests that a DLC be stopped. It is returned indicating whether the DLC was successfully stopped. STOP_DLC is also used to instruct the Program to stop automatically retrying the activation of any link stations on ports over this DLC.

VCB Structure

```
typedef struct stop_dlc
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

stop_type
Manner in which DLC should be stopped.

AP_ORDERLY_STOP
Node should perform cleanup operations before stopping DLC.

AP_IMMEDIATE_STOP
Node should stop DLC immediately.

dlc_name
Name of DLC to be stopped. This is an 8-byte string in a locally displayable character set, which must match that on the DEFINE_DLC verb.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLC

AP_UNRECOGNIZED_DEACT_TYPE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_STOP_DLC_PENDING

If the verb does not execute because it has been canceled, the Program returns the following parameter:

primary_rc
AP_CANCELLED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

STOP_INTERNAL_PU

The STOP_INTERNAL_PU verb requests the dependent LU requester (DLUR) initiate SSCP-PU session deactivation for a previously defined local PU that is served by DLUR.

VCB Structure

```
typedef struct stop_internal_pu
{
        unsigned short  opcode;         /* verb operation code    */
        unsigned char   reserv2;        /* reserved               */
        unsigned char   format;         /* format                 */
        unsigned short  primary_rc;     /* primary return code    */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
Name of the internal PU for which the SSCP-PU session will be deactivated. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

stop_type
Specifies stop type requested for the PU. An orderly stop will deactivate all underlying PLU-SLU and SSCP-LU sessions before deactivating the SSCP-PU session.

AP_ORDERLY_STOP
AP_IMMEDIATE_STOP

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_STOP_TYPE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PU_NOT_DEFINED

AP_PU_ALREADY_DEACTIVATING
AP_PU_NOT_ACTIVE

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

STOP_LS

STOP_LS requests the deactivation of a link station. It is returned specifying whether the link was stopped successfully. STOP_LS can also be used to disable remote activation of a link station or to disable activation on demand of a link station. STOP_LS is also used to instruct the Program to stop automatically retrying the activation of any link station.

VCB Structure

```
typedef struct stop_ls
{
        unsigned short  opcode;         /* verb operation code          */
        unsigned char   reserv2;        /* reserved                     */
        unsigned char   format;         /* format                       */
        unsigned short  primary_rc;     /* primary return code          */
        unsigned long   secondary_rc;   /* secondary return code        */
        unsigned char   stop_type;      /* stop type                    */
        unsigned char   ls_name[8];     /* name of link station         */
        unsigned char   disable;        /* whether the link is disabled */
        unsigned char   reserved[3];    /* reserved                     */
} STOP_LS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_STOP_LS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

stop_type
Manner in which the link station should be stopped.

AP_ORDERLY_STOP
Node should perform cleanup operations before stopping the link station.

AP_IMMEDIATE_STOP
Node should stop the link station immediately.

ls_name
Name of link station to be stopped. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. The value of **ls_name** must match that on the DEFINE_LS verb.

disable
This indicates whether remote activation or activation on demand of this link station should be disabled. If set to AP_NO, then the link station is returned to the state given by the values of **auto_act_supp** and **disable_remote_act** from the DEFINE_LS verb. Otherwise, the following values are possible (and can be ORed together).

AP_AUTO_ACT
The link cannot be reactivated on demand by the local node.

AP_REMOTE_ACT
The link cannot be activated by the remote node. For a link

configured with **disable_remote_act** set to AP_YES, this bit is ignored (activation by a remote node is always disabled by STOP_LS).

If the **disable** field is not set to AP_NO, then STOP_LS can be issued for a link that is not active or that is in the process of deactivating, for the purpose of setting the **disable** field.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_UNRECOGNIZED_DEACT_TYPE

AP_LINK_NOT_DEFD

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LINK_DEACT_IN_PROGRESS

If the verb does not execute because it was canceled, the Program returns the following parameter:

primary_rc
AP_CANCELLED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

STOP_PORT

STOP_PORT requests that a port be stopped. It is returned specifying whether the port was stopped successfully. STOP_PORT is also used to instruct the Program to stop automatically retrying the activation of any link stations on the port.

VCB Structure

```
typedef struct stop_port
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

stop_type
Manner in which the port should be stopped.

AP_ORDERLY_STOP
Node should perform cleanup operations before stopping the port.

AP_IMMEDIATE_STOP
Node should stop the port immediately.

port_name
Name of port to be stopped. This is an 8-byte string in a locally displayable character set, which must match that on the DEFINE_PORT verb.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PORT_NAME

AP_UNRECOGNIZED_DEACT_TYPE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_STOP_PORT_PENDING

If the verb does not execute because it has been canceled, the Program returns the following parameter:

primary_rc
AP_CANCELLED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

ACTIVATE_SESSION

The ACTIVATE_SESSION verb requests activation of a session between the local LU and a specified partner LU using the characteristic of a particular mode.

VCB Structure

Format 1

```
typedef struct activate_session
{
        unsigned short  opcode;         /* verb operation code     */
        unsigned char   reserv2;        /* reserved                */
        unsigned char   format;         /* format                  */
        unsigned short  primary_rc;     /* primary return code     */
        unsigned long   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 identifier      */
        unsigned char   cnos_permitted; /* is implicit CNOS        */
                                        /* permitted?              */
        unsigned char   reserv4[15];    /* reserved                */
} ACTIVATE_SESSION;
```

Format 0 (back-level)

```
typedef struct activate_session
{
        unsigned short  opcode;         /* verb operation code     */
        unsigned char   reserv2;        /* reserved                */
        unsigned char   format;         /* format                  */
        unsigned short  primary_rc;     /* primary return code     */
        unsigned long   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 identifier      */
} ACTIVATE_SESSION;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_ACTIVATE_SESSION

format
Identifies the format of the VCB. Set this field to zero or one to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to activate a session. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to activate a session. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_alias** and the **lu_name** are set to all zeros then the verb is forwarded to the LU associated with the control point (the default LU).

plu_alias
Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

mode_name
Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

fqplu_name
Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

polarity
The polarity requested for the session. Possible values are:

AP_POL_EITHER
AP_POL_FIRST_SPEAKER
AP_POL_BIDDER

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

cnos_permitted
This field may be set to AP_YES or AP_NO. If the activation of a new session is not possible because the session limits for the specified mode are reset, and this field is set to AP_YES, then the Program initiates implicit CNOS processing to initialize the session limits. Execution of this verb will be suspended while CNOS processing takes place.

Returned Parameters

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

primary_rc
AP_OK

secondary_rc
AP_AS_SPECIFIED

AP_AS_NEGOTIATED

session_id
8-byte identifier of the activated session.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_EXCEEDS_MAX_ALLOWED

AP_INVALID_CNOS_PERMITTED
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_MODE_NAME
AP_INVALID_PLU_NAME

If the verb exceeds the session limit for the mode, the Program returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

Secondary_rc
AP_EXCEEDS_MAX_ALLOWED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

If the verb does not execute because of other errors, the Program returns one of the following parameters:

primary_rc
AP_ACTIVATION_FAIL_NO_RETRY

AP_ACTIVATION_FAIL_RETRY

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 Node Operator Facility API, it is primarily intended for use by application programmers writing transaction programs that use the Personal Communications or Communications Server APPC API. The conversation group identifier is returned by the MC_ALLOCATE, ALLOCATE, MC_GET_ATTRIBUTES, GET_ATTRIBUTES and RECEIVE_ALLOCATE verbs defined in *Personal Communications for Windows, Version 5.7 Client/Server Communications Programming*.

VCB Structure

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

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEACTIVATE_CONV_GROUP

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to deactivate the conversation group. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to deactivate the conversation group. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and **lu_alias** are set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

conv_group_id
Conversation group identifier for the session to be deactivated.

type Type of deactivation. This field is a bitmask consisting of a deactivation type ORed with a flag indicating whether the verb should complete asynchronously or synchronously.

Deactivation types:

AP_DEACT_CLEANUP
The session is terminated immediately, without waiting for a response from the partner LU.

AP_DEACT_NORMAL
The session terminates after all conversations using the session are ended.

Verb behavior:

AP_ASYNCHRONOUS_DEACTIVATION
The verb returns immediately.

AP_SYNCHRONOUS_DEACTIVATION
The verb returns only after the session has been deactivated.

sense_data
Specifies the sense data for use in the CLEANUP type of deactivation.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CLEANUP_TYPE

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEACTIVATE_SESSION

The DEACTIVATE_SESSION verb requests the deactivation of a particular session, or all sessions on a particular mode.

VCB Structure

```
typedef struct deactivate_session
{
        unsigned short  opcode;         /* verb operation code     */
        unsigned char   reserv2;        /* reserved                */
        unsigned char   format;         /* format                  */
        unsigned short  primary_rc;     /* primary return code     */
        unsigned long   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                */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to deactivate a session. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to deactivate a session. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** fields are set to all zeros then the verb is forwarded to the LU associated with the control point (the default LU).

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

plu_alias
Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

mode_name
Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

type Type of deactivation. This field is a bitmask consisting of a deactivation type ORed with a flag indicating whether the verb should complete asynchronously or synchronously.

Deactivation types:

AP_DEACT_CLEANUP
The session is terminated immediately, without waiting for a response from the partner LU.

AP_DEACT_NORMAL
The session terminates after all conversations using the session are ended.

Verb behavior:

AP_ASYNCHRONOUS_DEACTIVATION
The verb returns immediately.

AP_SYNCHRONOUS_DEACTIVATION
The verb returns only after the session has been deactivated.

sense_data
Specifies the sense data to be used for the CLEANUP type of deactivation.

fqplu_name
Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

Returned Parameters

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

primary_rc
AP_OK

Note that if the **session_id** cannot be matched with any existing sessions, it is assumed that this is because the session has already been deactivated. In this case the verb completes successfully.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MODE_NAME

AP_INVALID_PLU_NAME
AP_INVALID_CLEANUP_TYPE
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

PATH_SWITCH

The PATH_SWITCH verb requests Personal Communications or Communications Server to switch routes on a connection that supports high-performance routing (HPR). If a better path cannot be found, the connection is left unchanged.

VCB Structure

```
typedef struct path_switch
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   secondary_rc;   /* secondary return code */
        unsigned char   rtp_connection_name[8];
                                        /* RTP connection name   */
} PATH_SWITCH;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_PATH_SWITCH

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

rtp_connection_name
Identifies the RTP connection to path-switch. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_RTP_CONNECTION

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PATH_SWITCH_IN_PROGRESS

If the verb does not execute because the path switch attempt fails, the Program returns the following parameter:

primary_rc
AP_UNSUCCESSFUL

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 6. Query Verbs

This chapter describes verbs used to query information about node configuration and status.

Only certain parameters are supported on SNA API clients.

QUERY_ADJACENT_NN

This verb applies only to Communications Server.

QUERY_ADJACENT_NN is only used at a network node and returns information about adjacent network nodes (that is, those network nodes to which CP-CP sessions are active or have been active or have been active at some time).

The adjacent node information is returned as a formatted list. To obtain information about a specific network node or to obtain the list information in several chunks, the **adj_nncp_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered on the **adj_nncp_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected the list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_adjacent_nn
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  total_num_entries; /* total number of entries      */
        unsigned char   list_options;      /* listing options              */
        unsigned char   reserv3;           /* reserved                     */
        unsigned char   adj_nncp_name[17]; /* CP name of adj network node  */
} QUERY_ADJACENT_NN;
typedef struct adj_nncp_data
{
        unsigned short  overlay_size;      /* size of this entry           */
        unsigned char   adj_nncp_name[17]; /* CP name of adj. network node */
        unsigned char   cp_cp_sess_status; /* CP-CP session status         */
        unsigned long   out_of_seq_tdus;   /* out of sequence TDUs         */
        unsigned long   last_frsn_sent;    /* last FRSN sent               */
        unsigned long   last_frsn_rcvd;    /* last FRSN received           */
        unsigned char   reserva[20];       /* reserved                     */
} ADJ_NNCP_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_ADJACENT_NN

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **adj_nncp_name** specified (see the following parameter, **adj_nncp_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

adj_nncp_name
Fully-qualified, 17 byte, name of adjacent network node composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

adj_nncp_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

adj_nncp_data.adj_nncp_name
17-byte fully-qualified CP name of adjacent network node which is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

adj_nncp_data.cp_cp_sess_status
Status of the CP-CP session. This is set to one of the following:

AP-ACTIVE
AP_CONWINNER_ACTIVE
AP_CONLOSER_ACTIVE
AP_INACTIVE

adj_nncp_data.out_of_seq_tdus
Number of out of sequence TDUs received from this node.

adj_nncp_data.last_frsn_sent
The last flow reduction sequence number sent to this node.

adj_nncp_data.last_frsn_rcvd
The last flow reduction sequence number received from this node.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_ADJ_NNCP_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_ADJACENT_NODE

QUERY_ADJACENT_NODE returns information about adjacent nodes configured on DEFINE_ADJACENT_NODE.

Information is returned in an ordered list. Each entry in the list consists of an ADJACENT_NODE_DATA overlay containing information about the adjacent CP, followed by an ADJACENT_NODE_LU_DATA overlay for each LU associated with the adjacent CP.

Entries are ordered by **cp_name**, then by **fqlu_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the list will start from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_adjacent_node
{
        unsigned short  opcode;            /* Verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* Primary return code       */
        unsigned long   secondary_rc;      /* Secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  total_num_entries; /* total number of entries   */
        unsigned char   list_options;      /* listing options           */
        unsigned char   reserv3;           /* reserved                  */
        unsigned char   cp_name[17];       /* CP name of adjacent node  */
} QUERY_ADJACENT_NODE;
typedef struct adjacent_node_data
{
        unsigned short  overlay_size;      /* size of this entry        */
        unsigned short  sub_overlay_size;  /* size of this stub entry   */
        unsigned char   cp_name[17];       /* CP name                   */
        DESCRIPTION     description;       /* resource description      */
        unsigned char   reserv3[19];       /* reserved                  */
        unsigned short  num_of_lus;        /* number of LUs             */
} ADJACENT_NODE_DATA;
typedef struct adjacent_node_lu_data
{
        unsigned short  overlay_size;      /* effective capacity        */
        unsigned char   reserve2[2];       /* reserved                  */
        ADJACENT_NODE_LU adj_lu_def_data;  /* Adjacent LU defined data  */
} ADJACENT_NODE_LU_DATA;
typedef struct adjacent_node_lu
{
        unsigned char   wildcard_lu;       /* Is this LU a wildcard?    */
        unsigned char   fqlu_name[17];     /* Fully-Qualified LU name   */
        unsigned char   reserve1[6];       /* reserved                  */
        ADJACENT_NODE_LU adj_lu_def_data;  /* Adjacent LU defined data  */
} ADJACENT_NODE_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_ADJACENT_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **cp_name** specified (see the following parameter, **cp_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first adjacent node in the directory maintained by the Program.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

cp_name
Fully qualified name of the adjacent node. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

adjacent_node_data.overlay_size
: The number of bytes in this entry, including any ADJACENT_NODE_LU_DATA structures, and hence the offset to the next entry returned (if any).

adjacent_node_data.sub_overlay_size
: The number of bytes in the node part of the entry, not including any ADJACENT_NODE_LU_DATA structures; this is the offset to the first ADJACENT_NODE_LU_DATA field in the entry.

adjacent_node_data.cp_name
: Fully qualified name of the adjacent node. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

adjacent_node_data.description
: Resource description (as specified on DEFINE_ADJACENT_NODE). The length of this field should be a multiple of four bytes, and nonzero.

adjacent_node_data.num_of_lus
: The number of LUs defined for this adjacent node. An ADJACENT_NODE_LU_DATA structure for each LU follows.

adjacent_node_lu_data.overlay_size
: The number of bytes in this entry, and hence the offset to the next entry returned (if any).

adjacent_node_lu.wildcard_lu
: Indicates whether the LU name is defined as a wildcard.

adjacent_node_lu.fqlu_name
: Fully qualified name of the adjacent node. The name is 17 bytes long and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

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

primary_rc
: AP_PARAMETER_CHECK

secondary_rc
: AP_INVALID_CP_NAME

: AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
: AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
: AP_UNEXPECTED_SYSTEM_ERROR

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).

The information is returned as a formatted list. To obtain information about a specific CN, or to obtain the list information in several chunks, the **fqcn_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered on the **fqcn_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the list will start from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_cn
{
        unsigned short  opcode;           /* Verb operation code       */
        unsigned char   attributes;       /* verb attributes           */
        unsigned char   format;           /* format                    */
        unsigned short  primary_rc;       /* Primary return code       */
        unsigned long   secondary_rc;     /* Secondary return code     */
        unsigned char   *buf_ptr;         /* pointer to buffer         */
        unsigned long   buf_size;         /* buffer size               */
        unsigned long   total_buf_size;   /* total buffer size required */
        unsigned short  num_entries;      /* number of entries         */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this 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
{
        unsigned short  num_act_ports;    /* number of active ports    */
        unsigned char   reserva[20];      /* reserved                  */
} CN_DET_DATA;

typedef struct cn_def_data
{
        unsigned char   description[RD_LEN];
                                          /* resource description      */
        unsigned char   num_ports;        /* number of ports on CN     */
        unsigned char   reserv1[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;       /* modem class               */
        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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **fqcn_name** specified (see the following parameter, **fqcn_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

fqcn_name
Fully qualified, 17-byte, connection network name. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

cn_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

cn_data.fqcn_name
Fully qualified, 17-byte, connection network name. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

cn_data.det_data.num_act_ports
Dynamic value giving number of active ports on the connection network.

cn_data.def_data.description
Resource description (as specified on DEFINE_CN). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

cn_data.def_data.num_ports
Number of ports on the connection network.

cn_data.def_data.tg_chars.effect_cap
Actual units of effective capacity. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 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. Valid values are integer values in the range 0–255, where 0 is the lowest cost per byte and 255 is the highest.

cn_data.def_data.tg_chars.security
: Security values as described in the list below.

  AP_SEC_NONSECURE
  : No security exists.

  AP_SEC_PUBLIC_SWITCHED_NETWORK
  : Data transmitted over this connection network will flow over a public switched network.

  AP_SEC_UNDERGROUND_CABLE
  : Data is transmitted over secure underground cable.

  AP_SEC_SECURE_CONDUIT
  : The line is a secure conduit that is not guarded.

  AP_SEC_GUARDED_CONDUIT
  : Conduit is protected against physical tapping.

  AP_SEC_ENCRYPTED
  : Encryption over the line.

  AP_SEC_GUARDED_RADIATION
  : Line is protected against physical and radiation tapping.

cn_data.def_data.tg_chars.prop_delay
: Propagation delay representing the time it takes for a signal to travel the length of the link, in microseconds. The value is encoded as a 1-byte floating-point number, represented by the formula 0.1mmm * 2 eeeee, where the bit representation of the byte is `eeeeemmm`. Default values are listed below.

  AP_PROP_DELAY_MINIMUM
  : No propagation delay.

  AP_PROP_DELAY_LAN
  : Less than 480 microseconds delay.

  AP_PROP_DELAY_TELEPHONE
  : Between 480 and 49 512 microseconds delay.

  AP_PROP_DELAY_PKT_SWITCHED_NET
  : Between 49 512 and 245 760 microseconds delay.

  AP_PROP_DELAY_SATELLITE
  : Longer than 245 760 microseconds delay.

  AP_PROP_DELAY_MAXIMUM
  : Maximum propagation delay.

cn_data.def_data.tg_chars.modem_class
: Reserved. This field should always be set to zero.

cn_data.def_data.tg_chars.user_def_parm_1
: User defined parameter in the range 0–255.

cn_data.def_data.tg_chars.user_def_parm_2
: User defined parameter in the range 0–255.

cn_data.def_data.tg_chars.user_def_parm_3
: User defined parameter in the range 0–255.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CN_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_CN_PORT

QUERY_CN_PORT returns information about ports defined on adjacent connection networks. The information is returned as a formatted list. To obtain information about a specific port, or to obtain the list information in several chunks, the **port_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. Note that the **fqcn_name** field must always be set to the name of a valid connection network.

See "Querying the Node" on page 10, for background on how the list formats are used.

VCB Structure

```
typedef struct query_cn_port
{
        unsigned short  opcode;            /* Verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* Primary return code       */
        unsigned long   secondary_rc;      /* Secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  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
{
        unsigned short  overlay_size;       /* size of this 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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The combination of **fqcn_name** and **port_name** specified (see the following parameters, **fqcn_name** and **port_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

fqcn_name
Fully qualified, 17-byte, connection network name. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field must always be set.

port_name
8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

cn_port_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

cn_port_data.fqcn_name
Fully qualified, 17-byte, connection network name. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

cn_port_data.port_name
Port name in an 8-byte, locally displayable character set. All 8 bytes are significant.

cn_port_data.tg_num
Transmission group number for specified port.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_CN_NAME

AP_INVALID_PORT_NAME
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_CONVERSATION

QUERY_CONVERSATION returns list information about conversations running over the specified LU. To obtain information about a specific conversation or to obtain the list information in several chunks, the **conv_id** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. Note that the **lu_alias** field must always be set. The lu_name, if nonzero, will be used in preference to the lu_alias.

See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **conv_id**. If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the index (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_conversation
{
        unsigned short  opcode;            /* Verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* Primary return code       */
        unsigned long   secondary_rc;      /* Secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  total_num_entries; /* total number of entries   */
        unsigned char   list_options;      /* listing options           */
        unsigned char   reserv3;           /* reserved                  */
        unsigned char   lu_name[8];        /* local LU name             */
        unsigned char   lu_alias[8];       /* local LU alias            */
        unsigned long   conv_id;           /* conversation identifier   */
        unsigned char   session_id[8];     /* session identifier        */
        unsigned char   reserv4[12];       /* reserved                  */
} QUERY_CONVERSATION;
```

```
typedef struct conv_summary
{
        unsigned short  overlay_size;      /* size of this entry        */
        unsigned long   conv_id;           /* conversation identifier   */
        unsigned char   local_tp_name[64]; /* Name of local TP          */
        unsigned char   partner_tp_name[64];
                                           /* Name of partner TP        */
        unsigned char   tp_id[8];          /* TP identifier             */
        unsigned char   sess_id[8];        /* session identifier        */
        unsigned long   conv_start_time;   /* time conversation was     */
                                           /* started                   */
        unsigned long   bytes_sent;        /* bytes sent so far         */
        unsigned long   bytes_received;    /* bytes received so far     */
        unsigned char   conv_state;        /* conversation state        */
        unsigned char   duplex_type;       /* conversation duplex type  */
} CONV_SUMMARY;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_CONVERSATION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **index** specified (see following) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
Name of the local LU. This is an 8–byte alphanumeric type A EBCDIC string (not starting with a number), and is right-padded with EBCDIC spaces.

lu_alias
Alias by which the local LU is known by the local TP. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

conv_id
Conversation ID.

session_id
If this is all binary zeroes, this field is not used to filter the returned conversations. If it is not zeroes, only those conversations whose session IDs match the supplied value are returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

conv_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

conv_summary.conv_id
Conversation ID.

The value of this parameter was returned by the ALLOCATE verb in the invoking transaction action or by RECEIVE_ALLOCATE in the invoked transaction program.

conv_summary.local_tp_name
Name of the local transaction program.

conv_summary.partner_tp_name
Name of the partner transaction program. This is only valid for a locally-initiated conversation. For a remotely-initiated conversation, it is blank.

conv_summary.tp_id
The transaction program identifier assigned to the transaction program. This identifier is either assigned by the API stub, or by the NOF transaction program manager.

conv_summary.sess_id
Identifier of the session allocated to this conversation.

conv_summary.conv_start_time
The elapsed time in centiseconds from the time the node was started to the time the conversation was started.

conv_summary.bytes_sent
The number of bytes sent so far on this conversation.

conv_summary.bytes_received
The number of bytes received so far on this conversation.

conv_summary_conv_state
Current state of the conversation that is identified by **conv_id**. For half-duplex conversations, it is one of the following:

AP_RESET_STATE

AP_SEND_STATE
AP_RECEIVE_STATE
AP_CONFIRM_STATE
AP_CONFIRM_SEND_STATE
AP_CONFIRM_DEALL_STATE
AP_PEND_POST_STAT
AP_PEND_DEALL_STATE

AP_END_CONV_STATE
AP_SEND_PENDING_STATE
AP_POST_ON_RECEIPT_STATE

For full-duplex conversations, it is one of the following:

AP_RESET_STATE
AP_SEND_RECEIVE_STATE
AP_SEND_ONLY_STATE
AP_RECEIVE_ONLY_STATE

conv_summary.duplex_type
Specifies whether this conversation is half or full-duplex.

AP_HALF_DUPLEX
AP_FULL_DUPLEX

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_INVALID_LU_ALIAS
AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_COS

QUERY_COS returns route calculation information for a specific class of service. The information is returned as a formatted list. To obtain information about a specific COS, or to obtain the list information in several chunks, the **cos_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used. This list is ordered on the **cos_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_cos
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   secondary_rc;     /* secondary return code        */
        unsigned char   *buf_ptr;         /* pointer to buffer            */
        unsigned long   buf_size;         /* buffer size                  */
        unsigned long   total_buf_size;   /* total buffer size required   */
        unsigned short  num_entries;      /* number of entries            */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this entry           */
        unsigned char   cos_name[8];      /* COS name                     */
        unsigned char   description[RD_LEN];
                                          /* resource description         */
        unsigned char   transmission_priority;
                                          /* transmission priority        */
        unsigned char   reserv1;          /* reserved                     */
        unsigned short  num_of_node_rows; /* number of node rows          */
        unsigned short  num_of_tg_rows;   /* number of TG rows            */
        unsigned long   trees;            /* number of tree caches for COS */
        unsigned long   calcs;            /* number of route calculations */
                                          /* for this COS                 */
        unsigned long   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **cos_name** specified (see the following parameter, **cos_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

cos_name
Class-of-service name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

cos_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

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

cos_data.description
Resource description (as specified on DEFINE_COS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

cos_data.transmission_priority
Transmission priority. This is set to one of the following values:

AP_LOW
AP_MEDIUM
AP_HIGH
AP_NETWORK

cos_data.num_of_node_rows
Number of node rows for this COS.

cos_data.num_of_tg_rows
Number of TG rows 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 (using the specified class of service) 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.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_COS_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DEFAULT_PU

QUERY_DEFAULT_PU allows the user to query the default PU defined using a DEFINE_DEFAULT_PU verb.

VCB Structure

```
typedef struct query_default_pu
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   secondary_rc;   /* secondary return code */
        unsigned char   def_pu_name[8]; /* default PU name       */
        unsigned char   description[RD_LEN];
                                        /* resource description  */
        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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

If the verb executes successfully, the Program 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 alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If no DEFINE_DEFAULT_PU verb has been issued then this field will be set to all zeros.

description
Resource description (as specified on DEFINE_DEFAULT_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_pu_sess
Name of the PU associated with the currently active default PU session. This will be different from the **def_pu_name** field if a default PU has been defined, but the session associated with it is not active. In this case, Personal Communications or Communications Server continues to use the session associated with the previous default PU until the session associated with the defined default PU becomes active. If there are no active PU sessions then this field will be set to all zeros.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DEFAULTS

QUERY_DEFAULTS allows the user to query the defaults defined using the DEFINE_DEFAULTS verb.

VCB Structure

```
typedef struct query_defaults
{
        unsigned short  opcode;         /* verb operation code   */
        unsigned char   reserv2;        /* reserved              */
        unsigned char   format;         /* format                */
        unsigned short  primary_rc;     /* primary return code   */
        unsigned long   secondary_rc;   /* secondary return code */
        DEFAULT_CHARS   default_chars;  /* default information   */
} QUERY_DEFAULTS;
```

```
typedef struct default_chars
{
        unsigned char   description[RD_LEN];
                                        /* resource description  */
        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           */
        unsigned char   limited_timeout;/* timeout for limited   */
                                        /* sessions              */
        unsigned char   reserv[244];    /* reserved              */
} DEFAULT_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DEFAULTS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

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

primary_rc
AP_OK

default_chars.description
Resource description (as specified on DEFINE_DEFAULTS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

default_chars.mode_name
Name of the mode specified on the most recent DEFINE_DEFAULTS verb. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If no DEFINE_DEFAULTS verb has been issued then this field will be set to all zeros.

default_chars.implicit_plu_forbidden
Controls whether the Program will put implicit definitions in place for unknown Partner LUs (AP_YES or AP_NO).

default_chars.specific_secuity_codes
Controls whether the Program will use specific sense codes on a security authentication or authorization failure (AP_YES or AP_NO). Note that the specific sense codes will only be returned to those partner LUs which have reported support for them on the session.

default_chars.limited_timeout
Specifies the timeout after which free limited-resource conwinnner sessions will be deactivated. Range 0 to 65535 seconds.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DIRECTORY_ENTRY

QUERY_DIRECTORY_ENTRY returns a list of LUs from the directory database. The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LU, or to obtain the list information in several chunks, the **resource_name** and **resource_type** fields should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

When the local node is a network node, information is returned as follows:

```
1st Network Node
                              1st LU located at Network Node
                              2nd LU locate at Network Node
                              ...
                              nth LU located at Network Node
              1st End Node served by this Network Node
                              1st LU located at End Node(1)
                              2nd LU located at End Node(1)
                              ...
                              nth LU located at End Node(1)
               ...
               nth End Node served by this Network Node
                              1st LU located at End Node(n)
                              2nd LU located at End Node(n)
                              ...
2nd Network Node
              ...etc..
```

When the Program is operating as an End Node the first entry returned in the first entry returned in the resource list is the EN CP. (No entry is returned for the End Node's Network Node server.)

This list of directory entries returned may be filtered by the parent name (and type). In this case, both the **parent_name** and **parent_type** fields should be set (otherwise these fields should be set to all zeros). Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

Format 1

```
typedef struct query_directory_entry{
        unsigned short  opcode;            /* verb operation code        */
        unsigned char   reserv2;           /* reserved                   */
        unsigned char   format;            /* format                     */
        unsigned short  primary_rc;        /* primary return code        */
        unsigned long   secondary_rc;      /* secondary return code      */
        unsigned char   *buf_ptr;          /* pointer to buffer          */
        unsigned long   buf_size;          /* buffer size                */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries          */
        unsigned short  total_num_entries; /* total number of entries    */
        unsigned char   list_options;      /* listing options            */
        unsigned char   reserv3;           /* reserved                   */
        unsigned char   resource_name[17];   /* network qualified res name  */
        unsigned char   reserv4;             /* reserved                    */
        unsigned short  resource_type;       /* Resource type               */
        unsigned char   parent_name[17];     /* parent name filter          */
        unsigned char   reserv5;             /* reserved                    */
        unsigned short  parent_type;         /* parent type                 */
        unsigned char   reserv6[24];         /* reserved                    */
 } QUERY_DIRECTORY_ENTRY;
```

```
typedef struct directory_entry_summary
{
        unsigned short  overlay_size;        /* size of this entry           */
        unsigned char   resource_name[17];   /* network qualified res name   */
        unsigned char   reserve1;            /* reserved                     */
        unsigned short  resource_type;       /* Resource type                */
        unsigned char   description[RD_LEN]; /* resource description         */
        unsigned char   real_owning_cp_type; /* real owning CP type          */
        unsigned char   real_owning_cp_name[17];
                                             /* real owning CP name          */
} DIRECTORY_ENTRY_SUMMARY;
```

```
typedef struct directory_entry_detail
{
        unsigned short  overlay_size;        /* size of this entry          */
        unsigned char   resource_name[17];   /* network qualified res name  */
        unsigned char   reserv1a;            /* reserved                    */
        unsigned short  resource type;       /* Resource type               */
        unsigned char   description[RD_LEN]; /* resource description        */
        unsigned char   parent_name[17];     /* network qualified           */
                                             /* parent name                 */
        unsigned char   reserv1b;            /* reserved                    */
        unsigned short  parent_type;         /* parent resource type        */
        unsigned char   entry_type;          /* Type of the directory entry */
        unsigned char   location;            /* Resource location           */
        unsigned char   real_owning_cp_type; /* real owning CP type         */
        unsigned char   real_owning_cp_name[17];                            */
                                             /* real owning CP name         */
        unsigned char   reserva;             /* reserved                    */
} DIRECTORY_LU_DETAIL;
```

VCB Structure

Format 0 (back-level)

```
typedef struct query_directory_entry{
        unsigned short  opcode;              /* verb operation code         */
        unsigned char   reserv2;             /* reserved                    */
        unsigned char   format;              /* format                      */
        unsigned short  primary_rc;          /* primary return code         */
        unsigned long   secondary_rc;        /* secondary return code       */
        unsigned char   *buf_ptr;            /* pointer to buffer           */
        unsigned long   buf_size;            /* buffer size                 */
        unsigned long   total_buf_size;      /* total buffer size required  */
        unsigned short  num_entries;         /* number of entries           */
        unsigned short  total_num_entries;   /* total number of entries     */
        unsigned char   list_options;        /* listing options             */
        unsigned char   reserv3;             /* reserved                    */
        unsigned char   resource_name[17];   /* network qualified res name  */
        unsigned char   reserv4;             /* reserved                    */
        unsigned short  resource_type;       /* Resource type               */
        unsigned char   parent_name[17];     /* parent name filter          */
        unsigned char   reserv5;             /* reserved                    */
        unsigned short  parent_type;         /* parent type                 */
} QUERY_DIRECTORY_ENTRY;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DIRECTORY_ENTRY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above. In addition to affecting the format of the VCB, only format 1 returns resources of AP_DLUR_LU_RESOURCE.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **resource_name** and **resource_type** specified (see the following parameters, **resource_name** and **resource_type**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

resource_type
Resource type. See one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE
AP_DLUR_LU_RESOURCE

This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

parent_name
Parent name filter. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If this field is set, then only directory entries belonging to the specified parent are returned (and in this case, the **parent_name** field must also be set). This field is if it is set to all zeros.

parent_type
The type of parent specified in the **parent_name** field. The type must be specified if the **parent_name** field is nonzero, otherwise this field should be set to zero. The can be set to one of the following:

AP_ENCP_RESOURCE
AP_NNCP_RESOURCE

This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of directory entries returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

directory_entry_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

directory_entry_summary.resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_entry_summary.resource_type
Resource type. This can be one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE
AP_DLUR_LU_RESOURCE

(Not returned if **format** is set to zero.)

directory_entry_summary.description
Resource description as specified on:

DEFINE_LOCAL_LU
DEFINE_DIRECTORY_ENTRY
DEFINE_ADJACENT_LEN_NODE or
DEFINE_ADJACENT_NODE

directory_entry_summary.real_owning_cp_type
NN and BrNN only: Real owning CP type. This can be one of the following:

AP_NONE
The real owning CP is a parent resource.

AP_ENCP_RESOURCE
The real owning CP is not the parent resource and is an EN.

Other node types: This field is set to AP_NONE.

directory_entry_summary.real_owning_cp_name
NN and BrNN only: Fully qualified real owning CP name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If the real owning CP is the parent, this field is set to binary zeroes.

If the real owning CP is not the parent, then this field is set to the name of the real owning CP.

The real owning CP is not the parent in the directory of the NNS of a BrNN if the resource is owned by an EN in the domain of the BrNN. In this case, the real owning CP is the EN, but the parent is the BrNN.

Other node types: This field is set to binary zeroes.

directory_entry_detail.overlay_size
The number of bytes in this entry, and therefore the offset to the next entry returned (if any).

directory_entry_detail.resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_entry_detail.resource_type
Resource type. This can be one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

directory_entry_detail.description
Resource description as specified on:

DEFINE_LOCAL_LU
DEFINE_DIRECTORY_ENTRY
DEFINE_ADJACENT_LEN_NODE or
DEFINE_ADJACENT_NODE

directory_entry_detail.parent_name
Fully-qualified parent name of the node serving the LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_entry_detail.parent_type
Parent resource type. This can be one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE

directory_entry_detail.entry_type
Specifies the type of the directory entry. This can be one of the following values:

AP_HOME
Local resource.

AP_CACHE
Cached entry.

AP_REGISTER
Registered resource (NN only).

directory_entry_detail.location
Specifies the location of the resource, which can be one of the following values:

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_entry_detail.real_owning_cp_type
NN and BrNN only: Real owning CP type. This can be one of the following:

AP_NONE
The real owning CP is a parent resource.

AP_ENCP_RESOURCE
The real owning CP is not the parent resource and is an EN.

Other node types: This field is set to AP_NONE.

directory_entry_detail.real_owning_cp_name
NN and BrNN only: Fully qualified real owning CP name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If the real owning CP is the parent, this field is set to binary zeroes.

If the real owning CP is not the parent, then this field is set to the name of the real owning CP.

The real owning CP is not the parent in the directory of the NNS of a BrNN if the resource is owned by an EN in the domain of the BrNN. In this case, the real owning CP is the EN, but the parent is the BrNN.

Other node types: This field is set to binary zeroes.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_RES_NAME

AP_INVALID_RES_TYPE
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DIRECTORY_LU

QUERY_DIRECTORY_LU returns a list of LUs from the directory database. The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LU, or to obtain the list information in several chunks, the **lu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **lu_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

Note that DLUS-served LUs present in the directory are also returned by this query.

VCB Structure

```
typedef struct query_directory_lu
{
        unsigned short  opcode;              /* verb operation code        */
        unsigned char   reserv2;             /* reserved                   */
        unsigned char   format;              /* format                     */
        unsigned short  primary_rc;          /* primary return code        */
        unsigned long   secondary_rc;        /* secondary return code      */
        unsigned char   *buf_ptr;            /* pointer to buffer          */
        unsigned long   buf_size;            /* buffer size                */
        unsigned long   total_buf_size;      /* total buffer size required */
        unsigned short  num_entries;         /* number of entries          */
        unsigned short  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
{
        unsigned short  overlay_size;        /* size of this entry         */
        unsigned char   lu_name[17];         /* network qualified LU name  */
        unsigned char   description[RD_LEN]; /* resource description       */
} DIRECTORY_LU_SUMMARY;
typedef struct directory_lu_detail
{
        unsigned short  overlay_size;        /* size of this entry         */
        unsigned char   lu_name[17];         /* network qualified LU name  */
        unsigned char   description[RD_LEN]; /* resource description       */
        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   apparent_lu_owner_name[17];
                                             /* apparent LU owner name     */
        unsigned char   reserva[3];          /* reserved                   */
} DIRECTORY_LU_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DIRECTORY_LU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **lu_name** specified (see the following parameter, **lu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
Network qualified LU name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of directory entries returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

directory_lu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

directory_lu_summary.lu_name
Network qualified LU name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_lu_summary.description
Resource description (as specified on DEFINE_LOCAL_LU, or DEFINE_ADJACENT_NODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

directory_lu_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

directory_lu_detail.lu_name
Network qualified LU name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_lu_detail.description
Resource description (as specified on DEFINE_LOCAL_LU, or DEFINE_ADJACENT_NODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

directory_lu_detail.server_name
Network qualified name of the node serving the LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_lu_detail.lu_owner_name
Network qualified name of the node owning the LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_lu_detail.location
Specifies the location of the resource, which can be one of the following values:

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 directory entry. This can be one of the following values:

AP_HOME
Local resource.

AP_CACHE
Cached entry.

AP_REGISTER
Registered resource (NN only).

directory_lu_detail.wild_card
Specifies the type of wildcard the LU will match.

AP_OTHER
Unknown type of LU entry.

AP_EXPLICIT
The full **lu_name** will be used for locating this LU.

AP_PARTIAL_WILDCARD
Only the nonspace portions of **lu_name** will be used for locating this LU.

AP_FULL_WILDCARD
All **lu_names** will be directed to this LU.

directory_lu_detail.apprent_lu_owner_name
NN and BrNN only: Fully qualified apparent LU owner CP name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If the apparent LU owner is the real LU owner, this field is set to binary zeroes.

If the apparent LU owner is not the real owner, then this field is set to the name of the apparent LU owner.

The real LU owner is not the apparent LU owner in the directory of the NNS of a BrNN if the resource is owned by an EN in the domain of the BrNN. In this case, the real LU owner is the EN, but the apparent owner is the BrNN.

Other node types: This field is set to binary zeroes.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DIRECTORY_STATS

This verb applies only to Communications Server.

QUERY_DIRECTORY_STATS returns directory database statistics. (The statistics that refer to cache information are reserved in the case of an end node). The verb can be used to gauge the level of network locate traffic. In the case of a network node this information can be used to tune the size of the directory cache, which is configurable at node-initialization time.

VCB Structure

```
typedef struct query_directory_stats
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   secondary_rc;     /* secondary return code        */
        unsigned long   max_caches;       /* max number of cache entries  */
        unsigned long   cur_caches;       /* cache entry count            */
        unsigned long   cur_home_entries; /* home entry count             */
        unsigned long   cur_reg_entries;  /* registered entry count       */
        unsigned long   cur_directory_entries;
                                          /* current number of dir entries */
        unsigned long   cache_hits;       /* count of cache finds         */
        unsigned long   cache_misses;     /* count of resources found by  */
                                          /* broadcast search (not cache) */
        unsigned long   in_locates;       /* locates in                   */
        unsigned long   in_bcast_locates; /* broadcast locates in         */
        unsigned long   out_locates;      /* locates out                  */
        unsigned long   out_bcast_locates; /* broadcast locates out       */
        unsigned long   not_found_locates; /* unsuccessful locates        */
        unsigned long   not_found_bcast_locates;
                                          /* unsuccessful broadcast       */
                                          /* locates                      */
        unsigned long   locates_outstanding;
                                          /* total outstanding locates    */
        unsigned char   reserva[20];      /* reserved                     */
} QUERY_DIRECTORY_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DIRECTORY_STATS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

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

primary_rc
AP_OK

max_caches
Reserved.

cur_caches
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
Reserved.

cache_misses
Reserved.

in_locates
Number of directed locates received.

in_bcast_locates
Number of broadcast locates received.

out_locates
Number of directed locates sent.

out_bcast_locates
Number of broadcast locates sent.

not_found_locates
Number of directed locates returned with a "not found."

not_found_bcast_locates
Number of broadcast locates returned with a "not found."

locates_outstanding
Current number of outstanding locates, both directed and broadcast.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DLC

QUERY_DLC returns a list of information about the DLCs defined at the node. This information is structured as determined data (data gathered dynamically during execution) and defined data (the data supplied by the application on DEFINE_DLC).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific DLC, or to obtain the list information in several chunks, the **dlc_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **dlc_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_dlc
{
        unsigned short  opcode;           /* verb operation code        */
        unsigned char   attributes;       /* ver attributes             */
        unsigned char   format;           /* format                     */
        unsigned short  primary_rc;       /* primary return code        */
        unsigned long   secondary_rc;     /* secondary return code      */
        unsigned char   *buf_ptr;         /* pointer to buffer          */
        unsigned long   buf_size;         /* buffer size                */
        unsigned long   total_buf_size;   /* total buffer size required */
        unsigned short  num_entries;      /* number of entries          */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this entry         */
        unsigned char   dlc_name[8];      /* name of DLC                */
        unsigned char   description[RD_LEN];
                                          /* resource description       */
        unsigned char   state;            /* State of the DLC           */
        unsigned char   dlc_type;         /* DLC type                   */
} DLC_SUMMARY;

typedef struct dlc_detail
{
        unsigned short  overlay_size;     /* size of this 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
{
       DESCRIPTION     description;       /* resource description       */
       unsigned char   dlc_type;          /* DLC type                   */
       unsigned char   neg_ls_supp;       /* negotiable LS support      */
       unsigned char   port_types;        /* allowable port types       */
       unsigned char   retry_flags;       /* conditions for automatic   */
                                          /* retries                    */
       unsigned short  max_activaion_attempts;
                                          /* how many automatic retries? */
       unsigned short  activation_delay_timer;
                                          /* delay between automatic    */
                                          /* retries                    */
       unsigned char   reserv3[6];        /* reserved                   */
       unsigned short  dlc_spec_data_len; /* Length of DLC specific data */
} DLC_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DLC

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **dlc_name** specified (see the following parameter, **dlc_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

dlc_name
DLC name. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

dlc_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

dlc_summary.dlc_name
DLC name. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

dlc_summary.description
Resource description (as specified on DEFINE_DLC). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

dlc_summary.state
State of the DLC. This field is set to one of the following values:

AP_ACTIVE
AP_NOT_ACTIVE
AP_PENDING_INACTIVE

dlc_summary.dlc_type
Type of DLC. The Program supports the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

dlc_detail.overlay_size
The number of bytes in this entry (including dlc_spec_data), and hence the offset to the next entry returned (if any).

dlc_detail.dlc_name
DLC name. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

dlc_detail.det_data.state
State of the DLC. This field is set to one of the following values:

AP_ACTIVE
AP_NOT_ACTIVE
AP_PENDING_INACTIVE

dlc_detail.def_data.description
Resource description (as specified on DEFINE_DLC). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

dlc_detail.def_data.dlc_type
Type of DLC. The Program supports the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

dlc_detail.def_data.neg_ls_supp
Specifies whether the DLC supports negotiable link stations (AP_YES or AP_NO).

dlc_detail.def_data.port_types
Specifies the allowable port types for the supplied **dlc_type**. The value corresponds to one or more of the following values ORed together:

AP_PORT_NONSWITCHED
AP_PORT_SWITCHED
AP_PORT_SATF

dlc__detail.def_data.retry_flags
This field specifies the conditions under which link stations, defined on this DLC, are subject to automatic retry if the flag AP_INHERIT_RETRY is set on both DEFINE_LS and DEFINE_PORT in **def_data.retry_flags**. It is a bit field, and may take any of the following values bitwise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, the Program will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, the Program attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_INHERIT_RETRY
This flag has no effect.

dlc_detail.def_data.max_activation_attempts
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags**, **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS, and **def_data.max_activation_attempts** on DEFINE_PORT is set to AP_USE_DEFAULTS.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means 'no limit'. The value AP_USE_DEFAULTS means 'no limit'.

dlc_detail.def_data.activation_delay_timer
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags**, **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS, and **def_data.max_activation_attempts** on DEFINE_PORT is set to AP_USE_DEFAULTS.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **def_data.retry_flags**.

The value of zero or AP_USE_DEFAULTS results in the use of default timer duration of thirty seconds.

dlc_detail.def_data.dlc_spec_data_len
Unpadded length, in bytes, of data specific to the type of DLC. The data will be concatenated to the DLC_DETAIL structure. This data will be padded to end on a 4-byte boundary. This field should always be set to zero.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLC_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DLUR_DEFAULTS

QUERY_DLUR_DEFAULTS allows the user to query the defaults defined using the DEFINE_DLUR_DEFAULTS verb.

VCB Structure

```
typedef struct query_dlur_defaults
{
        unsigned short  opcode;            /* verb operation code        */
        unsigned char   reserv2;           /* reserved                   */
        unsigned char   format;            /* format                     */
        unsigned short  primary_rc;        /* primary return code        */
        unsigned long   secondary_rc;      /* secondary return code      */
        DESCRIPTION     description;       /* resource description       */
        unsigned char   dlus_name[17];     /* DLUS name                   */
        unsigned char   bkup_dlus_name[17];/* Backup DLUS name             */
        unsigned char   reserv3;           /* reserved                   */
        unsigned short  dlus_retry timeout;/* DLUS Retry Timeout         */
        unsigned short  dlus_retry_limit;  /* DLUS Retry Limit           */
        unsigned char   reserv4[16];       /* reserved                   */
} QUERY_DLUR_LU;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DLUR_DEFAULTS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

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

primary_rc
AP_OK

description
Resource description. The length of this field should be a multiple of four bytes and nonzero.

dlus_name
Name of the DLUS node that will serve as the default. This is set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

bkup_dlus_name
Name of the DLUS node that will serve as the backup default. This is set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

dlus_retry_timeout
Interval in seconds between the 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. If X'FFFF' is specified, the Program retries indefinitely.

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameters:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the system has not been built with DLUR support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because a STOP_NODE verb has been issued, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DLUR_LU

QUERY_DLUR_LU returns a list of information about DLUR-supported LUs.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LU, or to obtain the list information in several chunks, the **lu_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered by the **lu_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The list of LUs returned can be filtered by **pu_name** or by whether the LU is local or downstream or by both. If filtering by PU is desired, the **pu_name** field should be set (otherwise this field should be set to all zeros). If filtering by location is desired, the **filter** field should be set to AP_INTERNAL or AP_DOWNSTREAM (otherwise, if no filtering is required, this field should be set to AP_NONE).

VCB Structure

```
typedef struct query_dlur_lu
{
        unsigned short  opcode;           /* verb operation code        */
        unsigned char   reserv2;          /* reserved                   */
        unsigned char   format;           /* format                     */
        unsigned short  primary_rc;       /* primary return code        */
        unsigned long   secondary_rc;     /* secondary return code      */
        unsigned char   *buf_ptr;         /* pointer to buffer          */
        unsigned long   buf_size;         /* buffer size                */
        unsigned long   total_buf_size;   /* total buffer size required */
        unsigned short  num_entries;      /* number of entries          */
        unsigned short  total_num_entries; /* total number of entries   */
        unsigned char   list_options;     /* listing options            */
        unsigned char   reserv3;          /* reserved                   */
        unsigned char   lu_name[8];       /* name of LU                 */
        unsigned char   pu_name[8];       /* name of PU to filter on    */
        unsigned char   filter;           /* reserved                   */
} QUERY_DLUR_LU;
typedef struct dlur_lu_summary
{
        unsigned short  overlay_size;     /* size of this entry         */
        unsigned char   lu_name[8];       /* name of LU                 */
} DLUR_LU_SUMMARY;
typedef struct dlur_lu_detail
{
        unsigned short  overlay_size;     /* size of this entry         */
        unsigned char   lu_name[8];       /* name of LU                 */
        unsigned char   pu_name[8];       /* 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;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DLUR_LU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **lu_name** specified (see the following parameter, **lu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
Name of LU being queried. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

pu_name
PU name filter. This should be set to all zeros or an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set then only LUs associated with the specified PU are returned. This field is ignored if it is set to all zeros.

filter Location filter. Specifies whether the returned LUs should be filtered by location (AP_INTERNAL or AP_DOWNSTREAM). If no filter is required, this field should be set to AP_NONE.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

dlur_lu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

dlur_lu_summary.lu_name
Name of LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlur_lu_detail.overlay_size
The number of bytes in this entry (including appended RSCV), and hence the offset to the next entry returned (if any).

dlur_lu_detail.lu_name
Name of LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlur_lu_detail.pu_name
Name of PU associated with the LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlur_lu_detail.dlus_name
Name of the DLUS node if the SSCP-LU session is active. This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the SSCP-LU session is not active, this field will be set to all zeros.

dlur_lu_detail.lu_location
Location of LU. The only value returned is:

AP_INTERNAL
AP_DOWNSTREAM

dlur_lu_detail.nau_address
Network addressable unit address of the LU. This is in the range 1–255.

dlur_lu_detail.plu_name
Name of PLU if the LU has an active PLU-SLU session. This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the PLU-SLU session is not active, this field will be set to all zeros.

dlur_lu_detail.rscv_len
This value will always be zero.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_FILTER_OPTION
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DLUR_PU

QUERY_DLUR_PU returns a list of information about DLUR-supported PUs.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific PU, or to obtain the list information in several chunks, the **pu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **pu_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The list of PUs returned can be filtered either by **dlus_name** or by whether the PU is local or downstream or by both. If filtering by DLUS is desired, the **dlus_name** field should be set (otherwise this field should be set to all zeros). If filtering by PU location is desired, the **filter** field should be set to AP_INTERNAL or AP_DOWNSTREAM (otherwise, if no filtering is required, this field should be set to AP_NONE).

VCB Structure

```
typedef struct query_dlur_pu
{
        unsigned short  opcode;             /* verb operation code          */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   format;             /* format                       */
        unsigned short  primary_rc;         /* primary return code          */
        unsigned long   secondary_rc;       /* secondary return code        */
        unsigned char   *buf_ptr;           /* pointer to buffer            */
        unsigned long   buf_size;           /* buffer size                  */
        unsigned long   total_buf_size;     /* total buffer size required   */
        unsigned short  num_entries;        /* number of entries            */
        unsigned short  total_num_entries;  /* total number of entries      */
        unsigned char   list_options;       /* listing options              */
        unsigned char   reserv3;            /* reserved                     */
        unsigned char   pu_name[8];         /* name of PU                   */
        unsigned char   dlus_name[17];      /* fully qualified DLUS name    */
        unsigned char   filter;             /* local/downstream filter      */
} QUERY_DLUR_PU;
typedef struct dlur_pu_summary
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   pu_name[8];         /* name of PU                   */
        unsigned char   description[RD_LEN];
                                            /* resource description         */
} DLUR_PU_SUMMARY;
typedef struct dlur_pu_detail
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   pu_name[8];         /* name of PU                   */
        unsigned char   description[RD_LEN];
                                            /* resource description         */
        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           */
        unsigned short  dlus_retry_timeout;  /* DLUS retry timeout            */
        unsigned short  dlus_retry_limit;    /* DLUS retry limit              */
} DLUR_PU_DETAIL;

typedef struct fqpcid
{
        unsigned char   pcid[8];             /* proc 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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **pu_name** specified (see the following parameter, **pu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

pu_name
Name of PU being queried. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dlus_name
DLUS filter. This should be set to all zeros or to a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. If this field is set then only PUs associated with an SSCP-PU session to the specified DLUS node are returned. This field is ignored if it is set to all zeros.

filter This field should be set to AP_NONE.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

dlur_pu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

dlur_pu_summary.pu_name
Name of PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlur_pu_summary.description
Resource description (as specified on DEFINE_INTERNAL_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

dlur_pu_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

dlur_pu_detail.pu_name
Name of PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

dlur_pu_detail.description
Resource description (as specified on DEFINE_INTERNAL_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

dlur_pu_detail.defined_dlus_name
Name of the DLUS node defined by either a DEFINE_INTERNAL_PU verb or DEFINE_LS verb (with **dspu_services** set to AP_DLUR). This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

dlur_pu_detail.bkup_dlus_name
Name of backup DLUS node defined by either a DEFINE_INTERNAL_PU verb or DEFINE_LS verb (with **dspu_services** set to AP_DLUR). This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

dlur_pu_detail.pu_id
PU identifier defined in a DEFINE_INTERNAL_PU verb or obtained in an XID from a downstream PU. This 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.

dlur_pu_detail.pu_location
Location of PU. The only value returned is:

AP_INTERNAL
AP_DOWNSTREAM

dlur_pu_detail.active_dlus_name
Name of the DLUS node that the PU is currently using. This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the SSCP-PU session is not active, this field will be set to all zeros.

dlur_pu_detail.ans_support
Auto Network Shutdown support. This field is reserved if the SSCP-LU session is inactive. The support setting is 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. This can be one of the following values:

AP_CONT
AP_STOP

dlur_pu_detail.pu_status
Status of the PU (as seen by DLUR). This can be set to one of the following values:

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. This can be one of the following values:

AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE
AP_INACTIVE

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 zeros.

dlur_pu_detail.fqpcid.fqcp_name
Fully qualified Control Point name used on the pipe. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the SSCP-PU session is not active this field will be set to zeros.

dlur_pu_detail.dlus_retry_timeout
Interval in seconds between second and subsequent attempts to contact the DLUS specified in the **dlus_name** and **bkup_dlus_name** fields. The interval between the initial attempt and the first retry is always one second. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used.

def_data.dlus_retry_limit
Maximum number of retries after an initial failure to contact the DLUS specified in the **dlus_name** and **bkup_dlus_name** fields. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. If X'FFFF' is specified, the Program retrys indefinitely.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_FILTER_OPTION
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DLUS

QUERY_DLUS returns a list of information about DLUS nodes known by DLUR.

The information is returned as a list. To obtain information about a specific DLUS node, or to obtain the list information in several chunks, the **dlus_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **dlus_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

Note that this verb returns pipe statistics.

VCB Structure

```
typedef struct query_dlus
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   secondary_rc;     /* secondary return code        */
        unsigned char   *buf_ptr;         /* pointer to buffer            */
        unsigned long   buf_size;         /* buffer size                  */
        unsigned long   total_buf_size;   /* total buffer size required   */
        unsigned short  num_entries;      /* number of entries            */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this 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       */
        unsigned short  num_active_pus;   /* num of active PUs using pipe */
        PIPE_STATS      pipe_stats;       /* pipe statistics              */
} DLUS_DATA;

typedef struct pipe_stats
{
        unsigned long   reqactpu_sent;    /* REQACTPUs sent to DLUS       */
        unsigned long   reqactpu_rsp_received;
                                          /* RSP(REQACTPU)s received      */
                                          /* from DLUS                    */
        unsigned long   actpu_received;   /* ACTPUs received from DLUS    */
        unsigned long   actpu_rsp_sent;   /* RSP(ACTPU)s sent to DLUS     */
        unsigned long   reqdactpu_sent;   /* REQDACTPUs sent to DLUS      */
        unsigned long   reqdactpu_rsp_received;
                                          /* RSP(REQDACTPU)s received     */
                                          /* from DLUS                    */
        unsigned long   dactpu_received;   /* DACTPUs received from DLUS    */
        unsigned long   dactpu_rsp_sent;   /* RSP(DACTPU)s sent to DLUS     */
        unsigned long   actlu_received;    /* ACTLUs received from DLUS     */
        unsigned long   actlu_rsp_sent;    /* RSP(ACTLU)s sent to DLUS      */
        unsigned long   dactlu_received;   /* DACTLUs received from DLUS    */
        unsigned long   dactlu_rsp_sent;   /* RSP(DACTLU)s sent to DLUS     */
        unsigned long   sscp_pu_mus_rcvd;  /* MUs for SSCP-PU               */
                                           /* sessions received             */
        unsigned long   sscp_pu_mus_sent;  /* MUs for SSCP-PU sessions sent */
        unsigned long   sscp_lu_mus_rcvd;  /* MUs for SSCP-LU sessions      */
                                           /* received                      */
        unsigned long   sscp_lu_mus_sent;  /* MUs for SSCP-LU sessions sent */
} PIPE_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DLUS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **dlus_name** specified (see the following parameter, **dlus_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

dlus_name
Name of the DLUS being queried. This should be set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC

spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

dlus_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

dlus_data.dlus_name
Name of the DLUS. This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

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. It can have one of the following values:

AP_ACTIVE
AP_PENDING_ACTIVE
AP_INACTIVE
AP_PENDING_INACTIVE

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.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DLUS_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DOWNSTREAM_LU

This verb applies only to Communications Server.

QUERY_DOWNSTREAM_LU returns information about downstream LUs served by DLUR or PU concentration or both. This information is structured as determined data (data gathered dynamically during execution) and defined data. (Defined data is supplied by the application on the DEFINE_DOWNSTREAM_LU verb. Note that for DLUR-supported LUs, implicitly defined data is put in place when the downstream LU is activated).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific local LU or to obtain the list information in several chunks, the **dslu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored.

The returned LUs may be filtered by the type of service the local node provides or the LU's associated downstream PU or both. If filtering by type of service is desired, the **dspu_services** field should be set to AP_PU_CONCENTRATION or AP_DLUR (otherwise, this field should be set to AP_NONE). If filtering by PU is desired, the **dspu_name** field should be set (otherwise, this field should be set to all zeros).

VCB Structure

```
typedef struct query_downstream_lu
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   attributes;       /* Verb attributes              */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   secondary_rc;     /* secondary return code        */
        unsigned char   *buf_ptr;         /* pointer to buffer            */
        unsigned long   buf_size;         /* buffer size                  */
        unsigned long   total_buf_size;   /* total buffer size required   */
        unsigned short  num_entries;      /* number of entries            */
        unsigned short  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;    /* filter on DSPU services type */
} QUERY_DOWNSTREAM_LU;
```

```
typedef struct downstream_lu_summary
{
        unsigned short  overlay_size;     /* size of this entry           */
        unsigned char   dslu_name[8];     /* LU name                      */
        unsigned char   dspu_name[8];     /* PU name                      */
        unsigned char   description[RD_LEN];
                                          /* resource description         */
        unsigned char   dspu_services;    /* type of service provided to  */
                                          /* downstream node              */
        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
{
        unsigned short  overlay_size;      /* size of this 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 services 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 sess stats   */
        unsigned char   host_lu_name[8];   /* Determined host LU name       */
        unsigned char   host_lu_name[8];   /* Determined host PU name       */
        unsigned char   reserva[4];        /* reserved                      */
} DOWNSTREAM_LU_DET_DATA;
typedef struct downstream_lu_def_data
{
        unsigned char   description[RD_LEN];
                                           /* resource description          */
        unsigned char   nau_address;       /* NAU address                   */
        unsigned char   dspu_name[8];      /* Downstream PU name            */
        unsigned char   host_lu_name;      /* 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;
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; /* max send BTU size             */
        unsigned short  max_rcv_btu_size;  /* max rcv BTU size              */
        unsigned short  max_send_pac_win;  /* max send pacing win size      */
        unsigned short  cur_send_pac_win;  /* current send pacing win size  */
        unsigned short  max_rcv_pac_win;   /* max receive pacing win 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 of FMD data frames sent   */
        unsigned long   send_data_bytes;   /* number of data bytes sent     */
        unsigned long   rcv_data_frames;   /* num data frames received      */
        unsigned long   rcv_fmd_data_frames;
                                           /* num of FMD data frames recvd  */
        unsigned long   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[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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **dslu_name** specified (see the following parameter, **dslu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

dslu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dspu_name
PU name filter. This should be set to all zeros or an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set, then only LUs associated with the specified PU are returned. This field is ignored if it is set to all zeros.

dspu_services
DSPU services filter. If set to AP_PU_CONCENTRATION, only

downstream LUs served by PU concentration are returned. If set to AP_DLUR, only DLUR-supported LUs are returned. Otherwise, if set to AP_NONE, information on all downstream LUs is returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

downstream_lu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

downstream_lu_summary.dslu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

downstream_lu_summary.dspu_name
Name of local PU that this LU is using. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

downstream_lu_summary.description
Resource description (as specified on DEFINE_DOWNSTREAM_LU or DEFINE_DOWNSTREAM_LU_RANGE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

downstream_lu_summary.dspu_services
Specifies the services which the local node provides to the downstream LU across the link. This is set to one of the following:

AP_PU_CONCENTRATION
Local node that provides PU concentration for the downstream LU.

AP_DLUR
Local node that provides DLUR support for the downstream LU.

downstream_lu_summary.nau_address
Network addressable unit address of the LU, which is in the range 1–255.

downstream_lu_summary.lu_sscp_sess_active
Indicates whether the LU-SSCP session is active (AP_YES or AP_NO).

downstream_lu_summary.plu_sess_active
Indicates whether the PLU-SLU session is active (AP_YES or AP_NO).

downstream_lu_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

downstream_lu_detail.dslu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

downstream_lu_detail.det_data.lu_sscp_sess_active
Indicates whether the LU-SSCP session to the downstream LU is active (AP_YES or AP_NO).

downstream_lu_detail.det_data.plu_sess_active
Indicates whether the PLU-SLU session to the downstream LU is active (AP_YES or AP_NO).

downstream_lu_detail.det_data.dspu_services
Specifies the services that the local node provides to the downstream LU across the link. This is set to one of the following values:

AP_PU_CONCENTRATION
Local node that provides PU concentration for the downstream LU.

AP_DLUR
Local node that provides DLUR support for the downstream LU.

downstream_lu_detail.det_data.lu_sscp_stats.rcv_ru_size
Maximum receive RU size. If **downstream_lu_detail.det_data.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.lu_sscp_stats.send_ru_size
Maximum send RU size. If **downstream_lu_detail.det_data.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.lu_sscp_stats.max_send_btu_size
Maximum BTU size that can be sent.

downstream_lu_detail.det_data.lu_sscp_stats.max_rcv_btu_size
Maximum BTU size that can be received.

downstream_lu_detail.det_data.lu_sscp_stats.max_send_pac_win
This field will always be set to zero.

downstream_lu_detail.det_data.lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

downstream_lu_detail.det_data.lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

downstream_lu_detail.det_data.lu_sscp_stats.cur_rcv_pac_win
This field will always be set to zero.

downstream_lu_detail.det_data.lu_sscp_stats.send_data_frames
Number of normal flow data frames sent.

downstream_lu_detail.det_data.lu_sscp_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

downstream_lu_detail.det_data.lu_sscp_stats.send_data_bytes
Number of normal flow data bytes sent.

downstream_lu_detail.det_data.lu_sscp_stats.rcv_data_frames
Number of normal flow data frames received.

downstream_lu_detail.det_data.lu_sscp_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

downstream_lu_detail.det_data.lu_sscp_stats.rcv_data_bytes
Number of normal flow data bytes received.

downstream_lu_detail.det_data.lu_sscp_stats.sidh
Session ID high byte.

downstream_lu_detail.det_data.lu_sscp_stats.sidl
Session ID low byte.

downstream_lu_detail.det_data.lu_sscp_stats.odai
Origin Destination Address 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_lu_detail.det_data.lu_sscp_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.

downstream_lu_detail.det_data.lu_sscp_stats.pacing_type
Receive pacing in use on the LU-SSCP session. This takes the value AP_NONE.

downstream_lu_detail.det_data.ds_plu_stats.rcv_ru_size
Maximum receive RU size.

downstream_lu_detail.det_data.ds_plu_stats.send_ru_size
Maximum send RU size.

downstream_lu_detail.det_data.ds_plu_stats.max_send_btu_size
Maximum BTU size that can be sent.

downstream_lu_detail.det_data.ds_plu_stats.max_rcv_btu_size
Maximum BTU size that can be received.

downstream_lu_detail.det_data.ds_plu_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

downstream_lu_detail.det_data.ds_plu_stats.cur_send_pac_win
Current size of the send pacing window on this session.

downstream_lu_detail.det_data.ds_plu_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

downstream_lu_detail.det_data.ds_plu_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

downstream_lu_detail.det_data.ds_plu_stats.send_data_frames
Number of normal flow data frames sent.

downstream_lu_detail.det_data.ds_plu_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

downstream_lu_detail.det_data.ds_plu_stats.send_data_bytes
Number of normal flow data bytes sent.

downstream_lu_detail.det_data.ds_plu_stats.rcv_data_frames
Number of normal flow data frames received.

downstream_lu_detail.det_data.ds_plu_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

downstream_lu_detail.det_data.ds_plu_stats.rcv_data_bytes
Number of normal flow data bytes received.

downstream_lu_detail.det_data.ds_plu_stats.sidh
Session ID high byte.

downstream_lu_detail.det_data.ds_plu_stats.sidl
Session ID low byte.

downstream_lu_detail.det_data.ds_plu_stats.odai
Origin Destination Address 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 1 if the BIND sender is the node containing the secondary link station.

downstream_lu_detail.det_data.ds_plu_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.

downstream_lu_detail.det_data.plu_stats.pacing_type
Receive pacing type in use on the downstream PLU-SLU session. This can take the values AP_NONE or AP_PACING_FIXED.

downstream_lu_detail.det_data.us_plu_stats.rcv_ru_size
Maximum receive RU size.

downstream_lu_detail.det_data.us_plu_stats.send_ru_size
Maximum send RU size.

downstream_lu_detail.det_data.us_plu_stats.max_send_btu_size
Maximum BTU size that can be sent.

downstream_lu_detail.det_data.us_plu_stats.max_rcv_btu_size
Maximum BTU size that can be received.

downstream_lu_detail.det_data.us_plu_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

downstream_lu_detail.det_data.us_plu_stats.cur_send_pac_win
Current size of the send pacing window on this session.

downstream_lu_detail.det_data.us_plu_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

downstream_lu_detail.det_data.us_plu_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

downstream_lu_detail.det_data.us_plu_stats.send_data_frames
Number of normal flow data frames sent.

downstream_lu_detail.det_data.us_plu_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

downstream_lu_detail.det_data.us_plu_stats.send_data_bytes
Number of normal flow data bytes sent.

downstream_lu_detail.det_data.us_plu_stats.rcv_data_frames
Number of normal flow data frames received.

downstream_lu_detail.det_data.us_plu_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

downstream_lu_detail.det_data.us_plu_stats.rcv_data_bytes
Number of normal flow data bytes received.

downstream_lu_detail.det_data.us_plu_stats.sidh
Session ID high byte. If **downstream_lu_detail.det_data_.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.us_plu_stats.sidl
Session ID low byte. If **downstream_lu_detail.det_data_.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.us_plu_stats.odai
Origin Destination Address 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 1 if the BIND sender is the node containing the secondary link station. If **downstream_lu_detail.det_data_.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.us_plu_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. If **downstream_lu_detail.det_data_dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_lu_detail.det_data.us_plu_stats.pacing_type
Receive pacing type in use on the upstream PLU-SLU session. This can take the values AP_NONE or AP_PACING_FIXED.

downstream_lu_detail.det_data.host_lu_name
Name of the host LU that the downstream LU is mapped to, or was mapped to when the PLU-SLU session was last active. This may differ from **def_data.host_lu_name**, as that may be the name of the host LU pool.

downstream_lu_detail.det_data.host_pu_name
Name of the host PU that the downstream PU is mapped to, or was mapped to when the PLU-SLU session was last active.

downstream_lu_detail.def_data.description
Resource description (as specified on DEFINE_DOWNSTREAM_LU or DEFINE_DOWNSTREAM_LU_RANGE).

downstream_lu_detail.def_data.nau_address
Network addressable unit address of the LU, which is in the range 1–255.

downstream_lu_detail.def_data.dspu_name
Name of PU associated with the LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

downstream_lu_detail.def_data.host_lu_name
Name of the host LU or host LU pool that the downstream LU is mapped to. In the case of an LU, this is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. In the case of an LU pool, the Program does not specify a character set for this field. This field is reserved for DLUR-served downstream LUs.

downstream_lu_detail.def_data.allow_timeout
Specifies whether the Program is allowed to time out host LUs used by this downstream LU if the session is left inactive for the **timeout** period specified on the host LU definition (AP_YES or AP_NO).

downstream_lu_detail.def_data.delayed_logon
Specifies whether the Program should delay connecting the downstream

LU to the host LU until the first data is received from the downstream LU. Instead, a simulated logon screen will be sent to the downstream LU (AP_YES or AP_NO).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DOWNSTREAM_PU

This verb applies only to Communications Server.

QUERY_DOWNSTREAM_PU returns information about downstream PUs (defined using a DEFINE_LS verb).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific local PU or to obtain the list information in several chunks, the **dspu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field is ignored.

The list of PUs can be filtered by the type of service the local node provides for the downstream PU. To do this, the **dspu_services** field should be set to AP_PU_CONCENTRATION or AP_DLUR.

VCB Structure

```
typedef struct query_downstream_pu
{
        unsigned short  opcode;            /* verb operation code           */
        unsigned char   attributes;        /* Verb attributes               */
        unsigned char   reserv2;           /* reserved                      */
        unsigned char   format;            /* format                        */
        unsigned short  primary_rc;        /* primary return code           */
        unsigned long   secondary_rc;      /* secondary return code         */
        unsigned char   *buf_ptr;          /* pointer to buffer             */
        unsigned long   buf_size;          /* buffer size                   */
        unsigned long   total_buf_size;    /* total buffer size required    */
        unsigned short  num_entries;       /* number of entries             */
        unsigned short  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            */
        unsigned char   dspu_services;     /* filter on DSPU services type  */
} QUERY_DOWNSTREAM_PU;
typedef struct downstream_pu_data
{
        unsigned short  overlay_size;      /* size of this entry            */
        unsigned char   dspu_name[8];      /* PU name                       */
        unsigned char   description[RD_LEN];
                                           /* resource description          */
        unsigned char   ls_name[8];        /* Link name                     */
        unsigned char   pu_sscp_sess_active;
                                           /* Is PU-SSCP session active     */
        unsigned char   dspu_services;     /* DSPU service type             */
        SESSION _STATS  pu_sscp_stats;     /* SSCP-PU session stats         */
        unsigned char   reserva[20];       /* reserved                      */
} DOWNSTREAM_PU_DATA
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; /* max send BTU size             */
        unsigned short  max_rcv_btu_size;  /* max rcv BTU size              */
        unsigned short  max_send_pac_win;  /* max send pacing win size      */
        unsigned short  cur_send_pac_win;  /* current send pacing win size  */
        unsigned short  max_rcv_pac_win;   /* max receive pacing win 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 of FMD data frames sent   */
        unsigned long   send_data_bytes;   /* number of data bytes sent     */
        unsigned long   rcv_data_frames;   /* num data frames received      */
        unsigned long   rcv_fmd_data_frames;
                                           /* num of FMD data frames recvd  */
        unsigned long   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[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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **dslu_name** specified (see the following parameter, **dslu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

dspu_name
Name of the downstream PU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dspu_services
DSPU services filter. If set to AP_PU_CONCENTRATION, only downstream LUs served by PU concentration are returned. If set to AP_DLUR, only DLUR-supported LUs are returned. Otherwise, if set to AP_NONE, information on all downstream LUs is returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

downstream_pu_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

downstream_pu_data.dspu_name
Name of the downstream PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

downstream_pu_data.description
Resource description (as specified on DEFINE_LS).

downstream_pu_data.ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

downstream_pu_data.pu_sscp_sess_active
Indicates whether the PU_SSCP session to the downstream PU is active. Set to either AP_YES or AP_NO.

downstream_pu_data.dspu_services
Specifies the services that the local node provides to the downstream PU across the link. This is set to one of the following values:

AP_PU_CONCENTRATION
Local node that provides PU concentration for the downstream LU.

AP_DLUR
Local node that provides DLUR support for the downstream LU.

downstream_pu_data.pu_sscp_stats.rcv_ru_size
Maximum receive RU size. If **downstream_lu_detail.det_data.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

downstream_pu_data.pu_sscp_stats.send_ru_size
Maximum send RU size. If **downstream_lu_detail.det_data.dspu_services** is set to AP_PU_CONCENTRATION, then this field is reserved.

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
This field will always be set to zero.

downstream_pu_data.pu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

downstream_pu_data.pu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

downstream_pu_data.pu_sscp_stats.cur_rcv_pac_win
This field will always be 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 Address 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 1 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 string in a locally displayable character set. All 8 bytes are significant.

downstream_pu_data.pu_sscp_stats.pacing_type
Receive pacing type in use on the upstream PU-SSCP session. This will take the value AP_NONE.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_TYPE
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_DSPU_TEMPLATE

This verb applies only to Communications Server.

QUERY_DSPU_TEMPLATE returns information about defined downstream PU templates used for PU concentration over implicit links. This information is returned as a list. To obtain information about a specific downstream PU template or to obtain the list information in several chunks, the **template_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field is ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

VCB Structure

```
typedef struct query_dspu_template
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   attributes;        /* Verb attributes              */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  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
{
        unsigned short  overlay_size;      /* size of this entry           */
        unsigned char   template_name[8];  /* name of DSPU template        */
        unsigned char   description;       /* resource description         */
        unsigned char   reserv1[12];       /* reserved                     */
        unsigned short  max_instance;      /* max active template instances */
        unsigned short  active instance;   /* current active instances     */
        unsigned short  num_of_dslu_templates;
                                           /* number of DSLU templates     */
} DSPU_TEMPLATE_DATA;
```

Each **dspu_template_data** is followed by **num_of_dslu_templates** downstream LU templates. Each downstream LU template has the following format.

```
typedef struct dslu_template_data
{
        unsigned short  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;           /* min NAU address in range     */
        unsigned char   max_nau;           /* max 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[10];       /* reserved                     */
        unsigned char   host_lu[8];        /* host LU or pool name         */
} DSLU_TEMPLATE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_DSPU_TEMPLATE

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

The **template_name** specified (see the following parameter, **template_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

template_name
Name of the DSPU template. This is an 8–byte string in a locally-displayable character set. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
: Number of entries actually returned.

total_num_entries
: Total number of entries that could have been returned. This can be higher than **num_entries**.

dspu_template_data.overlay_size
: The number of bytes in this entry (including any downstream LU templates, and hence the offset to the next entry returned, if any).

dspu_template_data.template_name
: Name of the DSPU template. This is an 8–byte string in a locally-displayable character set.

dspu_template_data.description
: Resource description (as specified on QUERY_DSPU_TEMPLATE).

dspu_template_data.max_instance
: This is the maximum number of instances of the template which can be active concurrently.

dspu_template_data.active_instance
: This is 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 field are **num_of_dslu_templates_application_id** entries, one for each application registered for the focal point category.

dslu_template_data.overlay_size
: The number of bytes in this entry (and hence the offset to the next entry returned, if any).

dslu_template_data.dslu_template.min_nau
: Minimum NAU address in the range.

dslu_template_data.dslu_template.max_nau
: Maximum NAU address in the range.

dslu_template_data.dslu_template.allow_timeout
: Specifies whether the Program is allowed to time out host LUs used by this downstream LU if the session is left inactive for the **timeout** period specified on the host LU definition (AP_YES or AP_NO).

dslu_template_data.dslu_template.delayed_logon
: Specifies whether the Program should delay 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 (AP_YES or AP_NO).

dslu_template_data.dslu_template.host_lu
: Name of the host LU or host LU pool that all the downstream LUs within the range will be mapped onto. This is an 8-byte alphanumeric type A-EBCDIC string (starting with a letter), padded to the right with EBCDIC Spaces.

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

primary_rc
: AP_PARAMETER_CHECK

secondary_rc
: AP_INVALID_TEMPLATE_NAME

: AP_INVALID_LIST_OPTION

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
: AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
: AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
: AP_UNEXPECTED_SYSTEM_ERROR

QUERY_FOCAL_POINT

QUERY_FOCAL_POINT returns information about focal points that Personal Communications or Communications Server knows about.

This information is returned as a list. To obtain information about a specific focal point category or to obtain the list information in several chunks, the **ms_category** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

Note: If no focal point is found, then one FP_DATA structure will be returned with **fp_data.fp_type** set to AP_NO_FP. See the following structure.

VCB Structure

```
typedef struct query_focal_point
{
        unsigned short  opcode;           /* verb operation code         */
        unsigned char   reserv2;          /* reserved                    */
        unsigned char   format;           /* format                      */
        unsigned short  primary_rc;       /* primary return code         */
        unsigned long   secondary_rc;     /* secondary return code       */
        unsigned char   *buf_ptr;         /* pointer to buffer           */
        unsigned long   buf_size;         /* buffer size                 */
        unsigned long   total_buf_size;   /* total buffer size required  */
        unsigned short  num_entries;      /* number of entries           */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this entry          */
        unsigned char   ms_appl_name[8];  /* focal point application name */
        unsigned char   ms_category[8];   /* focal point category        */
        unsigned char   description[RD_LEN];
                                          /* resource description        */
        unsigned char   fp_fqcp_name[17]; /* focal pt fully qual CP name */
        unsigned char   bkup_appl_name[8]; /* backup focal pt appl name  */
        unsigned char   bkup_fp_fqcp_name[17];
                                          /* backup FP fully qualified   */
                                          /* CP name                     */
        unsigned char   implicit_appl_name[8];
                                          /* implicit FP 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   reserva[20];      /* reserved                    */
        unsigned short  number_of_appls;  /* number of applications      */
} FP_DATA;
```

Each **fp_data** is followed by **number_of_appls** application names. Each application name has the following format:

```
typedef struct application_id
{
        unsigned char    appl_name[8];     /* application name             */
} APPLICATION_ID;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_FOCAL_POINT

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **ms_category** specified (see the following parameter, **ms_category**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

ms_category
Management services category. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services categories as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

fp_data.overlay_size
The number of bytes in this entry (including any application names, and hence the offset to the next entry returned (if any)).

fp_data.ms_appl_name
Name of the currently active focal point application. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services applications as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name.

fp_data.ms_category
Management services category. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services categories as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name.

fp_data.description
Resource description (as specified on DEFINE_FOCAL_POINT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

fp_data.fp_fqcp_name
Currently active focal point's fully qualified control point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

fp_data.bkup_appl_name
Name of backup focal point application. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services applications as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name.

fp_data.bkup_fp_fqcp_name
Backup focal point's fully qualified control point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

fp_data.implicit_appl_name
Name of implicit focal point application (specified using the DEFINE_FOCAL_POINT verb). This can either be one of the four byte architecturally defined values (right-padded with EBCDIC spaces) for management services applications as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name. This field will be the same as the **ms_appl_name** if the implicit focal point is the currently active focal point.

fp_data.implicit_fp_fqcp_name
Implicit focal point's fully qualified control point name (as specified using the DEFINE_FOCAL_POINT verb). This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field will be the same as the **fp_fqcp_name** if the implicit focal point is the currently active focal point.

fp_data.fp_type
Type of focal point. Refer to *SNA Management Services* for further detail. This will be one of the following values:

AP_EXPLICIT_PRIMARY_FP
AP_BACKUP_FP
AP_DEFAULT_PRIMARY_FP
AP_IMPLICIT_PRIMARY_FP
AP_DOMAIN_FP
AP_HOST_FP
AP_NO_FP

fp_data.fp_status
Status of the focal point. This can be one of the following values:

AP_NOT_ACTIVE
The focal point is currently not active.

AP_ACTIVE
The focal point is currently 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
Type of routing that applications should specify when using MDS transport to send data to the focal point.

AP_DEFAULT
Default routing is used to deliver the MDS_MU to the focal point.

AP_DIRECT
The MDS_MU will be routed on a session directly to the focal point.

fp_data.number_of_appls
Number of applications registered for this focal point category. Following this field will be **number_of_appls application_id entries**, one for each application registered for the focal point category.

application_id.appl_name
Name of application registered for focal point category. This can either be one of the 4-byte architecturally defined values (right-padded with

EBCDIC spaces) for management services applications as described in SNA management services, or an 8-byte type 1134 EBCDIC installation defined name.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MS_CATEGORY

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_HPR_STATS

This verb applies only to Communications Server.

QUERY_HPR_STATS returns statistics describing the HPR performance of the node. QUERY_HPR_STATS is only supported by nodes that support the RTP Tower.

VCB Structure

```
typedef struct query_hpr_stats
{
        unsigned short  opcode;             /* verb operation code        */
        unsigned char   reserv2;            /* reserved                   */
        unsigned char   format;             /* format                     */
        unsigned short  primary_rc;         /* primary return code        */
        unsigned long   secondary_rc;       /* secondary return code      */
        unsigned COUNTER
                        num-orig_rs_sent;   /* RS requests sent as origin */
        unsigned COUNTER
                        num_orig_rs_rej;    /* RS rejections at orign     */
        unsigned COUNTER
                        num_inter_rs_rcvd;  /* Intermediate RS requests   */
        unsigned COUNTER
                        num_inter_rs_rej;   /* Intermediate RS rejections */
        unsigned COUNTER
                        num_dest_rs_rcvd;   /* RS reqs as destination     */
        unsigned COUNTER
                        num_dest_rs_rej;    /* RS rej sent as destination */
        unsigned long   active_isr_hpr_sessions;
                                            /* ISR sessions active        */
        unsigned char   reserv[28];         /* reserved                   */
} QUERY_HPR_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_HPR_STATS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

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

primary_rc
AP_OK

num_orig_rs_sent
The total number HPR Route Setup requests sent that originated in this node, since the node started.

num_orig_rs_rej
The total number of HPR Route Setup requests that originated in this node and have been rejected by other nodes since the node started.

num_inter_rs_rcvd
The total number of HPR Route Setup requests processed by this node acting as an intermediate node since the node started.

num_inter_rs_rej
The total number of HPR Route Setup requests processed by this node acting as an intermediate node, that have been rejected by the node since the node started.

num_dest_rs_rcvd
The total number of HPR Route Setup requests received by this node, that has this node as the destination, since the node started.

num_dest_rs_rej
The total number of HPR Route Setup requests received by this node, that has this node as the destination and that have been rejected by the node since the node started.

active_isr_hpr_sessions
The number of ISR sessions using HPR-APPN Boundary Function that are currently active in the node.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node does not support the HPR RTP Tower function, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

QUERY_ISR_SESSION

This verb applies only to Communications Server.

QUERY_ISR_SESSION is only used at a Network Node and returns list information about sessions for which the network node is providing intermediate session routing.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific session, or to obtain the list information in several chunks, the fields in the **fqpcid** structure should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), the fields in this structure is ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered by **fqpcid.pcid** first and then by EBCDIC lexicographical ordering on **fqpcid.fqcp_name**. The ordering by **fqpcid.pcid_name** is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM’s 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The format of the **fqpcid** structure is an 8-byte Procedure Correlator Identifier (PCID) and the network qualified CP name of the session originator.

In addition to the detail information for each session, a route selection control vector (RSVC) is returned if this is specified on the START_NODE parameters. This RSVC defines the route through the network that the session takes in a hop-by-hop form.

VCB Structure

Format 2

```
typedef struct query_isr_session
{
   unsigned short  opcode;            /* verb operation code         */
   unsigned char   reserv2;           /* reserved                    */
   unsigned char   format;            /* format                      */
   unsigned short  primary_rc;        /* primary return code         */
   unsigned long   secondary_rc;      /* secondary return code       */
   unsigned char   *buf_ptr;          /* pointer to buffer           */
   unsigned long   buf_size;          /* buffer size                 */
   unsigned long   total_buf_size;    /* total buffer size required  */
   unsigned short  num_entries;       /* number of entries           */
   unsigned short  total_num_entries; /* total number of entries     */
   unsigned char   list_options;      /* listing options             */
   unsigned char   session_type;      /* is this query for DLUR or   */
                                      /* regular ISR sessions?       */
   FQPCID          fqpcid;            /* fully qualified procedure   */
                                      /* correlator ID               */
} QUERY_ISR_SESSION;

typedef struct isr_session_summary
{
   unsigned short  overlay_size;      /* size of this entry          */
   FQPCID          fqpcid;            /* fully qualified procedure   */
                                      /* correlator ID               */
} ISR_SESSION_SUMMARY;
```

```
typedef struct isr_session_detail
{
   unsigned short  overlay_size;      /* size of this entry           */
   FQPCID          fqpcid;            /* fully qualified procedure    */
   unsigned short  sub_overlay_size;  /* offset to appended RSCV      */
                                      /* correlator ID                */
   unsigned char   trans_pri;         /* Transmission priority:       */
   unsigned char   cos_name[8];       /* Class-of-service name        */
   unsigned char   ltd_res;           /* Session spans a limited      */
   unsigned char   reserv1[8];        /* reserved                    */
                                      /* resource                     */
   SESSION_STATS   pri_sess_stats;    /* primary hop session stats    */
   SESSION_STATS   sec_sess_stats;    /* secondary hop session        */
                                      /* statistics                   */
   unsigned char   sess_lu_type;      /* session LU type              */
   unsigned char   sess_lu_level;     /* session LU level             */
   unsigned char   pri_tg_number;     /* Primary session TG number    */
   unsigned char   sec_tg_number;     /* Secondary session TG number  */
   unsigned long   rtp_tcid;          /* RTP TC identifier            */
   unsigned long   time_active;       /* time elapsed since           */
                                      /* activation                   */
   unsigned char   isr_state;         /* current state of ISR session */
   unsigned char   reserv2[11];       /* reserved                     */
   unsigned char   mode_name[8];      /* mode name                    */
   unsigned char   pri_lu_name[17];   /* primary LU name              */
   unsigned char   sec_lu_name[17];   /* secondary LU name            */
   unsigned char   pri_adj_cp_name[17];
                                      /* primary stage adj CP name    */
   unsigned char   sec_adj_cp_name[17];
                                      /* secondary stage adj CP name  */
   unsigned char   reserv3[3];        /* reserved                     */
   unsigned char   rscv_len;          /* Length of following RSCV     */
} ISR_SESSION_DETAIL;

typedef struct fqpcid
{
   unsigned char   pcid[8];           /* pro correlator identifier    */
   unsigned char   fqcp_name[17];     /* orig's network qualified     */
                                      /* CP name                      */
   unsigned char   reserve3[3];       /* reserved                     */
} FQPCID;

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;  /* Max send pacing window size  */
   unsigned short  cur_send_pac_win;  /* Curr send pacing window size */
   unsigned short  max_rcv_pac_win;   /* Max receive pacing win size  */
   unsigned short  cur_rcv_pac_win;   /* Curr rec pacing window size  */
   unsigned long   send_data_frames;  /* Number of data frames sent   */
   unsigned long   send_fmd_data_frames;
                                      /* num of FMD data frames sent  */
   unsigned long   send_data_bytes;   /* Number of data bytes sent    */
   unsigned long   rcv_data_frames;   /* Num data frames received     */
   unsigned long   rcv_fmd_data_frames;
                                      /* num of FMD data frames recvd */
   unsigned long   rcv_data_bytes;    /* Num 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[8];        /* Link station name            */
   unsigned char   pacing_type;       /* type of pacing in use        */
} SESSION_STATS;
```

VCB Structure

Format 1 (back-level)

```
typedef struct isr_session_detail
{
   unsigned short  overlay_size;      /* size of this entry           */
   FQPCID          fqpcid;            /* fully qualified procedure    */
   unsigned short  sub_overlay_size;  /* offset to appended RSCV      */
                                      /* correlator ID                */
   unsigned char   trans_pri;         /* Transmission priority:       */
   unsigned char   cos_name[8];       /* Class-of-service name        */
   unsigned char   ltd_res;           /* Session spans a limited      */
   unsigned char   reserv1[2];        /* reserved                    */
                                      /* resource                     */
   SESSION_STATS   pri_sess_stats;    /* primary hop session stats    */
   SESSION_STATS   sec_sess_stats;    /* secondary hop session        */
                                      /* statistics                   */
   unsigned char   sess_lu_type;      /* session LU type              */
   unsigned char   sess_lu_level;     /* session LU level             */
   unsigned char   pri_tg_number;     /* Primary session TG number    */
   unsigned char   sec_tg_number;     /* Secondary session TG number  */
   unsigned long   rtp_tcid;          /* RTP TC identifier            */
   unsigned long   time_active;       /* time elapsed since           */
                                      /* activation                   */
   unsigned char   isr_state;         /* current state of ISR session */
   unsigned char   reserv2[11];       /* reserved                     */
   unsigned char   mode_name[8];      /* mode name                    */
   unsigned char   pri_lu_name[17];   /* primary LU name              */
   unsigned char   sec_lu_name[17];   /* secondary LU name            */
   unsigned char   pri_adj_cp_name[17];
                                      /* primary stage adj CP name    */
   unsigned char   sec_adj_cp_name[17];
                                      /* secondary stage adj CP name  */
   unsigned char   reserv3[3];        /* reserved                     */
   unsigned char   rscv_len;          /* Length of following RSCV     */
} ISR_SESSION_DETAIL;

typedef struct fqpcid
{
   unsigned char   pcid[8];           /* pro correlator identifier    */
   unsigned char   fqcp_name[17];     /* orig's network qualified     */
                                      /* CP name                      */
   unsigned char   reserve3[3];       /* reserved                     */
} FQPCID;

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;  /* Max send pacing window size  */
   unsigned short  cur_send_pac_win;  /* Curr send pacing window size */
   unsigned short  max_rcv_pac_win;   /* Max receive pacing win size  */
   unsigned short  cur_rcv_pac_win;   /* Curr rec pacing window size  */
   unsigned long   send_data_frames;  /* Number of data frames sent   */
   unsigned long   send_fmd_data_frames;
                                      /* num of FMD data frames sent  */
   unsigned long   send_data_bytes;   /* Number of data bytes sent    */
   unsigned long   rcv_data_frames;   /* Num data frames received     */
   unsigned long   rcv_fmd_data_frames;
                                      /* num of FMD data frames recvd */
   unsigned long   rcv_data_bytes;    /* Num 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[8];        /* Link station name            */
   unsigned char   pacing_type;       /* type of pacing in use        */
} SESSION_STATS;
```

VCB Structure

Format 0 (back-level)

```
typedef struct isr_session_detail
{
   unsigned short  overlay_size;      /* size of this entry           */
   FQPCID          fqpcid;            /* fully qualified procedure    */
   unsigned char   trans_pri;         /* Transmission priority:       */
   unsigned char   cos_name[8];       /* Class-of-service name        */
   unsigned char   ltd_res;           /* Session spans a limited      */
   unsigned char   reserv1[8];        /* reserved                     */
                                      /* resource                     */
   SESSION_STATS   pri_sess_stats;    /* primary hop session stats    */
   SESSION_STATS   sec_sess_stats;    /* secondary hop session        */
                                      /* statistics                   */
   unsigned char   reserv3[3];        /* reserved                     */
   unsigned char   reserva[20];       /* reserved                     */
   unsigned char   rscv_len;          /* Length of following RSCV     */
} ISR_SESSION_DETAIL;
```

Note: The ISR session detail overlay 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. The inclusion of this RSCV is decided when the node is started (as an option of the START_NODE), and can be altered later using DEFINE_ISR_STATS. If these verbs have been used to specify that RSCVs should not be stored, then the **rscv_len** is set to zero.

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_ISR_SESSION

format
Identifies the format of the VCB and also the format of the returned overlays. Set this field to zero to specify the version of the VCB and overlays listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **fqpcid** specified (see the following parameter, **fqpcid**) represent an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

session_type
Does this verb query DLUR-maintained sessions, or regular ISR sessions?

AP_ISR_SESSION	ISR sessions
AP_DLUR_SESSIONS	DLUR sessions

fqpcid.pcid
Procedure Correlator ID. This is an 8-byte hexadecimal string. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

fqpcid.fqcp_name
Fully qualified Control Point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

isr_session_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

isr_session_summary.fqpcid.pcid
Procedure Correlator ID.

isr_session_summary.fqpcid.fqcp_name
Fully qualified Control Point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

isr_session_detail.overlay_size
The number of bytes in this entry (including any appended RSCV), and hence the offset to the next entry returned (if any).

isr_session_detail.sub_overlay_size
This field gives the size of this detail overlay. If an RSCV is appended, then this is the offset to the start of the RSCV. This field can be equal to or greater than the size of the format of one detail structure (allowing future expansion).

isr_session_detail.fqpcid.pcid
Procedure Correlator ID.

isr_session_detail.fqpcid.fqcp_name
Fully qualified Control Point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

session_detail.trans_pri
Transmission priority. This is set to one of the following values:

AP_LOW
AP_MEDIUM
AP_HIGH
AP_NETWORK

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

session_detail.ltd_res
Specifies whether the session uses a limited resource link (AP_YES or AP_NO).

isr_session_detail.pri_sess_stats.rcv_ru_size
Maximum receive RU size.

isr_session_detail.pri_sess_stats.send_ru_size
Maximum send RU size.

isr_session_detail.pri_sess_stats.max_send_btu_size
Maximum BTU size that can be sent on primary session hop.

isr_session_detail.pri_sess_stats.max_rcv_btu_size
Maximum BTU size that can be received on the primary session hop.

isr_session_detail.pri_sess_stats.max_send_pac_win
Maximum size of the send pacing window on the primary session hop.

isr_session_detail.pri_sess_stats.cur_send_pac_win
Current size of the send pacing window on the primary session hop.

isr_session_detail.pri_sess_stats.max_rcv_pac_win
Maximum size of the receive pacing window on the primary session hop.

isr_session_detail.pri_sess_stats.cur_rcv_pac_win
Current size of the receive pacing window on the primary session hop.

isr_session_detail.pri_sess_stats.send_data_frames
Number of normal flow data frames sent on the primary session hop.

isr_session_detail.pri_sess_stats.send_data_frames
Number of normal flow data frames sent on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.send_data_bytes
Number of normal flow data bytes sent on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.rcv_data_frames
Number of normal flow data frames received on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.rcv_data_bytes
Number of normal flow data bytes received on the primary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.pri_sess_stats.sidh
Session ID high byte.

isr_session_detail.pri_sess_stats.sidl
Session ID low byte.

isr_session_detail.pri_sess_stats.odai
Origin destination address 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.

isr_session_detail.pri_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 data flows.

isr_session_detail.sec_sess_stats.rcv_ru_size
Maximum receive RU size.

isr_session_detail.pri_sess_stats.pacing_type
Receive pacing type in use on primary session. This may take the values AP_NONE, AP_PACING_FIXED or AP_PACING_ADAPTIVE.

isr_session_detail.sec_sess_stats.send_ru_size
Maximum send RU size.

isr_session_detail.sec_sess_stats.max_send_btu_size
Maximum BTU size that can be sent on secondary session hop.

isr_session_detail.sec_sess_stats.max_rcv_btu_size
Maximum BTU size that can be received on the secondary session hop.

isr_session_detail.sec_sess_stats.max_send_pac_win
Maximum size of the send pacing window on the secondary session hop.

isr_session_detail.sec_sess_stats.cur_send_pac_win
Current size of the send pacing window on the secondary session hop.

isr_session_detail.sec_sess_stats.max_rcv_pac_win
Maximum size of the receive pacing window on the secondary session hop.

isr_session_detail.sec_sess_stats.cur_rcv_pac_win
Current size of the receive pacing window on the secondary session hop.

isr_session_detail.sec_sess_stats.send_data_frames
Number of normal flow data frames sent on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.send_data_bytes
Number of normal flow data bytes sent on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.rcv_data_frames
Number of normal flow data frames received on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.rcv_data_bytes
Number of normal flow data bytes received on the secondary session hop. Zero will be returned in this field if collection of statistics has been disabled using DEFINE_ISR_STATS.

isr_session_detail.sec_sess_stats.sidh
Session ID high byte.

isr_session_detail.sec_sess_stats.sidl
Session ID low byte (from LFSID).

isr_session_detail.sec_sess_stats.odai
Origin destination address 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.

isr_session_detail.sec_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 intermediate session statistics with a particular link station.

isr_session_detail.sec_sess_stats.pacing_type
Receive pacing type in use on primary session. This can take the values AP_NONE, AP_PACING_FIXED, or AP_PACING_ADAPTIVE..

isr_session_detail.sess_lu_type
The LU type of the session specified on the BIND. This field takes one of the following values:

AP_LU_TYPE_0
AP_LU_TYPE_1
AP_LU_TYPE_2
AP_LU_TYPE_3
AP_LU_TYPE_4
AP_LU_TYPE_6
AP_LU_TYPE_7
AP_LU_TYPE_UNKNOWN
(LU type 5 is intentionally omitted.)

AP_LU_TYPE_UNKNOWN will always be returned unless collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.sess_lu_level
The LU level of the session. This field takes one of the following values:

AP_LU_LEVEL_0
AP_LU_LEVEL_1
AP_LU_LEVEL_2
AP_LU_LEVEL_UNKNOWN

For LU types other than 6, this field is set to AP_LU_LEVEL_0. AP_LU_LEVEL_UNKNOWN will always be returned unless collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.pri_tg_number
The TG number associated with the link traversed by the primary session hop. If the primary session stage traverses an RTP connection, zero is returned. Zero will always be returned unless collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.sec_tg_number
The TG number associated with the link traversed by the primary session hop. If the primary session stage traverses an RTP connection, zero is returned. Zero will always be returned unless collection of names has been enabled using DEINE_ISR_STATS.

isr_session_detail.rtp_tcid
The local TC ID for the RTP connection, returned in cases where this ISR session forms part of an ANR/ISR boundary. In other cases, this field is set to zero. Zero will always be returned unless collection of names has been enabled using DEINE_ISR_STATS.

isr_session_detail.time_active
The elapsed time since the activation of the session, measured in hundredths of a second. Zero will always be returned unless collection of names has been enabled using DEINE_ISR_STATS.

isr_session_detail.isr_state
The current state of the session. This field is set to one of the following values:

AP_ISR_INACTIVE
AP_ISR_PENDING_ACTIVE
AP_ISR_ACTIVE
AP_ISR_PENDING_INACTIVE

isr_session_detail.mode_name
The mode name for the session. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. All binary zeros will always be returned unless collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.pri_lu_name
The primary LU name of the session. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. Each name can have a maximum of 8 bytes with no embedded spaces. If this name is not available, all binary zeros are returned in this field. All binary zeros will always be returned unless a collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.sec_lu_name
The secondary LU name of the session. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. Each name can have a maximum of 8 bytes with no embedded spaces. If this name is not available, all binary zeros are returned in this field. All binary zeros will always be returned unless a collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.pri_adj_cp_name
The primary stage adjacent CP name of this session. If the primary session stage traverses an RTP connection, the CP name of the remote RTP endpoint is returned. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. Each name can have a maximum of 8 bytes with no embedded spaces. If this name is not available, all binary zeros are returned in this field. All binary zeros will always be returned unless a collection of names has been enabled using DEFINE_ISR_STATS.

isr_session_detail.sec_adj_cp_name
The secondary stage adjacent CP name of this session. If the secondary session stage traverses an RTP connection, the CP name of the remote RTP endpoint is returned. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. Each name can have a maximum of 8 bytes with no embedded spaces. If this name is not available, all binary zeros are returned in this field. All binary zeros will always be returned unless a collection of names has been enabled using DEFINE_ISR_STATS.

isr_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.) The RSCV will be padded to end on a 4–byte boundary.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_FQPCID

AP_INVALID_LIST_OPTION
AP_INVALID_SESSION_TYPE

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the node has not been built with network node support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_LOCAL_LU

QUERY_LOCAL_LU returns information about local LUs. QUERY_LOCAL_LU can be issued to retrieve information about the Personal Communications or Communications Server control point LU.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific local LU, or to obtain the list information in several chunks, the **lu_name** or **lu_alias** field should be set. If the **lu_name** field is nonzero it will be used to determine the index. If the **lu_name** field is set to all zeros, the **lu_alias** will be used to determine the index. If both the **lu_name** and the **lu_alias** fields are set to all zeros then the LU associated with the control point (the default LU) will be used. If the **list_options** field is set to AP_FIRST_IN_LIST then both of these fields will be ignored. (In this case, the returned list will be ordered by LU alias if the AP_LIST_BY_ALIAS **list_options** is set, otherwise it will be ordered by LU name). See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered on either **lu_alias** or **lu_name** according to the options specified. The field is ordered by EBCDIC lexicographical ordering.

The list of local LUs returned can be filtered by the name of the PU that they are associated with. In this case, the **pu_name** field should be set (otherwise this field should be set to all zeros).

VCB Structure

Format 1

```
typedef struct query_local_lu
{
        unsigned short  opcode;            /* verb operation code            */
        unsigned char   reserv2;           /* reserved                       */
        unsigned char   format;            /* format                         */
        unsigned short  primary_rc;        /* primary return code            */
        unsigned long   secondary_rc;      /* secondary return code          */
        unsigned char   *buf_ptr;          /* pointer to buffer              */
        unsigned long   buf_size;          /* buffer size                    */
        unsigned long   total_buf_size;    /* total buffer size required     */
        unsigned short  num_entries;       /* number of entries              */
        unsigned short  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
{
        unsigned short  overlay_size;      /* size of this entry             */
        unsigned char   lu_name[8];        /* LU name                        */
        unsigned char   lu_alias[8];       /* LU alias                       */
        unsigned char   description;       /* resource description           */
} LOCAL_LU_SUMMARY;
typedef struct local_lu_detail
{
        unsigned short  overlay_size;      /* size of this 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[RD_LEN];
                                   /* resource description             */
        unsigned char   lu_alias[8];       /* local LU alias                   */
        unsigned char   nau_address;       /* NAU address                      */
        unsigned char   syncpt_support;    /* Reserved                         */
        unsigned short  lu_session_limit;  /* LU session limit                 */
        unsigned char   default_pool;      /* member of default_lu_pool        */
        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       */
        unsigned char   attach_routing_data[128];
                                   /* routing data for                 */
                                   /* incoming attaches                */
        unsigned char   lu_model;          /* LU model name for SDDLU          */
        unsigned char   model_name[8];     /* LU model name for SDDLU          */
        unsigned char   reserv4[16];       /* reserved                         */
} 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; /* Is LU-SSCP session active         */
        unsigned char   reserv1[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
{
        unsigned short  rcv_ru_size;       /* session receive RU size          */
        unsigned short  send_ru_size;      /* session send RU size             */
        unsigned short  max_send_btu_size; /* max send BTU size                */
        unsigned short  max_rcv_btu_size;  /* max rcv BTU size                 */
        unsigned short  max_send_pac_win;  /* max send pacing win size         */
        unsigned short  cur_send_pac_win;  /* current send pacing win size     */
        unsigned short  max_rcv_pac_win;   /* max receive pacing win 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 of FMD data frames sent      */
        unsigned long   send_data_bytes;   /* number of data bytes sent        */
        unsigned long   rcv_data_frames;   /* num data frames received         */
        unsigned long   rcv_fmd_data_frames;
                                   /* num of FMD data frames recvd     */
        unsigned long   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[8];        /* Link station name                */
        unsigned char   pacing _type;      /* Type of pacing in use            */
} SESSION_STATS;
```

VCB Structure

Format 0

```
typedef struct local_lu_def_data
{
        unsigned char   description[RD_LEN];
                                   /* resource description             */
        unsigned char   lu_alias[8];       /* local LU alias                   */
        unsigned char   nau_address;       /* NAU address                      */
        unsigned char   syncpt_support;    /* Reserved                         */
        unsigned short  lu_session_limit;  /* LU session limit              */
        unsigned char   default_pool;      /* member of default_lu_pool     */
        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    */
        unsigned char   attach_routing_data[128];
                                           /* routing data for              */
                                           /* incoming attaches             */
} LOCAL_LU_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_LOCAL_LU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros) specified represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

AP_LIST_BY_ALIAS
The returned list is ordered by **lu_alias**. This option is only valid when AP_FIRST_IN_LIST is specified. If AP_LIST_FROM_NEXT or AP_LIST_INCLUSIVE is specified, the list ordering will depend on whether an **lu_name** or **lu_alias** has been supplied as a starting point.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** field are set to all zeros, the LU associated with the control point (the default LU) is used. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

pu_name
PU name filter. This should be set to all zeros or an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set then only Local LUs associated with this PU are returned. This field is ignored if it is set to all zeros.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

local_lu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

local_lu_summary.lu_name
LU name. This name is an 8-byte type-A EBCDIC character string.

local_lu_summary.lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

local_lu_summary.description
Resource description (as specified on DEFINE_LOCAL_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

local_lu_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

local_lu_detail.lu_name
LU name. This name is an 8-byte type-A EBCDIC character string.

local_lu_detail.def_data.description
Resource description (as specified on DEFINE_LOCAL_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

local_lu_detail.def_data.lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

local_lu_detail.def_data.nau_address
Network addressable unit address of the LU, which is in the range 0–255. A nonzero value implies the LU is a dependent LU. Zero implies the LU is an independent LU.

local_lu_detail.def_data.syncpt_support
Reserved.

local_lu_detail.def_data.lu_session_limit
Maximum number of sessions for the local LU. A value of zero indicates that there is no limit.

local_lu_detail.def_data.default_pool
AP_YES if the LU is a member of the dependent LU 6.2 default pool. Always AP_NO for independent LUs.

local_lu_detail.def_data.pu_name
Name of the PU that this LU will use. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is only used by dependent LUs, and will be set to all binary zeros for independent LUs.

local_lu_detail.def_data.lu_attributes
Configured LU attributes. This field either takes the value AP_NONE, or the following option ORed together:

AP_DISABLE_PWSUB
Password substitution support disabled for the local LU.

local_lu_detail.def_data.sscp_id
This field specifies the ID of the SSCP permitted to activate this LU. It is a 6–byte binary field. This field is only used by dependent LUs, and should be set to all binary zeros for independent LUs or if the LU can be activated by any SSCP.

local_lu_detail.def_data.disable
This field indicates whether the LOCAL LU should be disabled or enabled. The LU can be dynamically enabled or disabled by re-issuing the DEFINE_LOCAL_LU with this parameter set as appropriate (AP_YES or AP_NO). When a disabled LU is enabled, the Program issues a NOTIFY (online). When an enabled LU is disabled, the Program issues a NOTIFY (off-line). If the LU is bound when it is disabled, then the Program issues an UNBIND followed by a NOTIFY (offline).

local_lu_detail.def_data.attach_routing_data
This field indicates data passed out unchanged on a DYNAMIC_LOAD_INDICATION resulting from attaches arriving for the transaction program at this local LU. For example, this field may be used to set a path to the transaction program’s working directory.

local_lu_detail.def_data.lu_model
Model type and number of the LU. This field is only used by dependent

LUs and should be set to AP_UNKNOWN for independent LUs. For dependent LUs, this is set to one of the following values:

AP_3270_DISPLAY_MODEL_2
AP_3270_DISPLAY_MODEL_3
AP_3270_DISPLAY_MODEL_4
AP_3270_DISPLAY_MODEL_5
AP_RJE_WKSTN
AP_PRINTER
AP_SCS_PRINTER
AP_UNKNOWN

For dependent LUs, if **model_name** is not set to all binary zeros, then this field is ignored. If a value other than AP_UNKNOWN is specified and the host system supports SDDLU (Self-Defining Dependent LU), the node will generate an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector will contain the machine type and model number corresponding to the value of this field. This field may be changed dynamically by re-issuing the verb. Changes will not come into effect until after the LU is closed and deactivated.

local_lu_detail.def_data.model_name
Model name of the LU. This field is only used by dependent LUs and should be set to binary zeros for independent LUs.

If this field is not set to binary zeros and the host system supports SDDLU, the node generates an unsolicited PSID NMVT reply in order to dynamically define the local LU at the host. The PSID subvector contains the name supplied in this field. The field may be changed dynamically reissuing the verb. Changes do not come into effect until after the LU is closed and deactivated.

local_lu_detail.det_data.lu_sscp_session_active
Specifies whether the LU-SSCP session is active (AP_YES or AP_NO). If the **def_data.nau_address** is zero, then this field is reserved.

local_lu_detail.det_data.appl_conn_active
Specifies whether an application is using the LU (AP_YES or AP_NO). If the **def_data.nau_address** is zero, then this field is reserved.

local_lu_detail.det_data.lu_sscp_stats.rcv_ru_size
This field is always reserved.

local_lu_detail.det_data.lu_sscp_stats.send_ru_size
This field is always reserved.

local_lu_detail.det_data.lu_sscp_stats.max_send_btu_size
Maximum 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 field will always be set to zero.

local_lu_detail.det_data.lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

local_lu_detail.det_data.lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

local_lu_detail.det_data.lu_sscp_stats.cur_rcv_pac_win
This field will always be 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 FMD data 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 Address Indicator. When bringing up a session, the sender of the ACTLU sets this field 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 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 this session with the link over which the session flows.

Note: The LU-SSCP statistics (**local_lu_detail.det_data.lu_sscp_stats**) are valid only when **nau_address** is not zero. Otherwise the fields are reserved.

local_lu_detail.det_data.lu_sscp_stats.pacing_type
Receive pacing type in use on the LU-SSCP session. This will be set to AP_NONE.

local_lu_detail.det_data.sscp_id
This is a 6–byte field containing the SSCP ID received in the ACTPU for the PU used by this LU.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_ALIAS

AP_INVALID_LU_NAME
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_LOCAL_TOPOLOGY

All APPN nodes maintain a local topology database that holds information about the transmission groups (TGs) to all adjacent nodes.

QUERY_LOCAL_TOPOLOGY allows information about these TGs to be returned.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific local TG, or to obtain the list information in several chunks, the **dest**, **dest_type**, and **tg_num** fields should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), these fields will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used. This list is ordered on **dest** first, then on **dest_type** and finally on **tg_num**. The **dest** name is ordered by name length first, then by lexicographical ordering for names of the same length. The **dest_type** field follows the order: AP_LEN_NODE, AP_NETWORK_NODE, AP_END_NODE, AP_VRN. The **tg_num** is ordered numerically.

If AP_LIST_INCLUSIVE is selected, the returned list starts from the first valid record of that name.

If AP_LIST_FROM_NEXT is selected, the list will begin from the first valid record with a name following the one specified.

VCB Structure

```
typedef struct query_local_topology
{
        unsigned short  opcode;            /* verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* primary return code       */
        unsigned long   secondary_rc;      /* secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  total_num_entries; /* total number of entries   */
        unsigned char   list_options;      /* listing options           */
        unsigned char   reserv3;           /* reserved                  */
        unsigned char   dest[17];          /* TG destination node       */
        unsigned char   dest_type;         /* TG destination node type  */
        unsigned char   tg_num;            /* TG number                 */
} QUERY_LOCAL_TOPOLOGY;
typedef struct local_topology_summary
{
        unsigned short  overlay_size;      /* size of this entry        */
        unsigned char   dest[17];          /* TG destination node       */
        unsigned char   dest_type;         /* TG destination node type  */
        unsigned char   tg_num;            /* TG number                 */
} LOCAL_TOPOLOGY_SUMMARY;
typedef struct local_topology_detail
{
        unsigned short  overlay_size;      /* size of this entry        */
        unsigned char   dest[17];          /* TG destination node       */
        unsigned char   dest_type;         /* TG destination node type  */
        unsigned char   tg_num;            /* TG number                 */
        unsigned char   reserv1;           /* reserved                  */
        LINK_ADDRESS    dlc_data;          /* DLC signalling data       */
        unsigned long   rsn;               /* resource sequence number  */
        unsigned char   status;            /* TG status                 */
        TG_DEFINED_CHARS tg_chars;         /* TG characteristics        */
        unsigned char   cp_cp_session_active;
                                           /* CP-CP session is active   */
        unsigned char   branch_link_type;  /* branch link type          */
        unsigned char   branch_tg;         /* TG is a branch TG         */
        unsigned char   reserva[13];       /* reserved                  */
} LOCAL_TOPOLOGY_DETAIL;

typedef struct link_address
{
        unsigned short  length;            /* length                    */
        unsigned short  reserve1;          /* reserved                  */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                           /* address                   */
} LINK_ADDRESS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_LOCAL_TOPOLOGY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **dest**, **dest_type** and **tg_num** specified (see the following parameters, **dest**, **dest_type**, and **tg_num**) represents an index value that is used to specify the starting point of the actual information to be returned .

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

dest
Fully qualified destination node name for the TG. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dest_type
Node type of the destination node for this TG. This can be one of the following values:

AP_NETWORK_NODE
AP_VRN
AP_END_NODE
AP_LEARN_NODE

If the **dest_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

tg_num
Number associated with the TG. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

local_topology_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

local_topology_summary.dest
Fully qualified destination node name for the TG. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

local_topology_summary.dest_type
Type of the destination node for this TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN
AP_END_NODE

Note that if **dest_type** is set to AP_END_NODE, this specifies that the TG destination is either to a LEN node or to an end node.

local_topology_summary.tg_num
Number associated with the TG.

local_topology_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

local_topology_detail.dest
Fully qualified destination node name for the TG. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

local_topology_detail.dest_type
Type of the destination node for this TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN
AP_END_NODE

Note that if **dest_type** is set to AP_END_NODE, this specifies that the TG destination is either to a LEN node or to an end node.

local_topology_detail.tg_num
Number associated with the TG.

local_topology_detail.dlc_data.length
Length of DLC address of connection to a VRN (set to zero if **dest_type** is not AP_VRN).

local_topology_detail.dlc_data.address
DLC address of connection to VRN.

local_topology_detail.rsn
Resource Sequence Number. This is assigned by the network node that owns this resource.

local_topology_detail.status
Specifies the status of the TG. This can be one or more of the following values ORed together:

AP_TG_OPERATIVE
AP_TG_CP_CP_SESSIONS
AP_TG_QUIESCING
AP_TG_HPR
AP_TG_RTP
AP_NONE

local_topology_detail.tg_chars
TG characteristics (See “DEFINE_CN” on page 31).

local_topology_detail.cp_cp_session_active
Specifies whether the local node’s contention winner CP-CP session is active (AP_NO or AP_YES).

local_topology_detail.branch_link_type
BrNN only. This branch link type of this TG. This is set to one of the following:

AP_UPLINK
This link is an uplink.

AP_DOWNLINK
The link is a downlink to an EN.

AP_DOWNLINK_TO_BRNN
The TG is a downlink to a BrNN that is showing its EN face.

AP_OTHERLINK
This link is an otherlink.

Other node types: This field is not meaningful and is always set to AP_BRNN_NOT_SUPPORTED.

local_topology_detail.branch_tg
NN only. Specifies whether the TG is a branch TG.

AP_NO
The TG is not a branch TG.

AP_YES
The TG is a branch TG.

Other node types: This field is not meaningful and is always set to AP_NO.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TG

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

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) and defined data (the data supplied by the application on DEFINE_LS).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LS, or to obtain the list information in several chunks, the **ls_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **ls_name**. Ordering is according to name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The list of link stations returned can be filtered by the name of the port that they belong to. In this case, the **port_name** field should be set (otherwise this field should be set to all zeros).

VCB Structure

Format 1

```
typedef struct query_ls
{
        unsigned short  opcode;             /* verb operation code           */
        unsigned char   attributes;         /* Verb attributes               */
        unsigned char   format;             /* format                        */
        unsigned short  primary_rc;         /* Primary return code           */
        unsigned long   secondary_rc;       /* Secondary return code         */
        unsigned char   *buf_ptr;           /* pointer to buffer             */
        unsigned long   buf_size;           /* buffer size                   */
        unsigned long   total_buf_size;     /* total buffer size required    */
        unsigned short  num_entries;        /* number of entries             */
        unsigned short  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];       /* name of link station          */
} QUERY_LS;
```

```
typedef struct ls_summary
{
        unsigned short  overlay_size;       /* size of this entry            */
        unsigned char   ls_name[8];         /* link station name             */
        unsigned char   description[RD_LEN];
                                            /* resource description          */
        unsigned char   dlc_type;           /* DLC type                      */
        unsigned char   state;              /* link station state            */
        unsigned short  act_sess_count;     /* currently active sess count   */
        unsigned char   det_adj_cp_name[17];
                                            /* determined adj CP name        */
        unsigned char   det_adj_cp_type;    /* determined adj 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
{
       unsigned short  overlay_size;       /* size of this entry            */
       unsigned char   ls_name[8];         /* link stations name            */
       LS_DET_DATA     det_data;           /* determined data               */
       LS_DEF_DATA     def_data;           /* defined data                  */
} LS_DETAIL;

typedef struct ls_det_data
{
       unsigned short  act_sess_count;     /* curr 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;            /* is LS is dynamic ?            */
       unsigned char   migration;          /* supports migration partners   */
       unsigned char   tg_num;             /* TG number                     */
       LS_STATS        ls_stats;           /* link station statistics       */
       unsigned long   start_time;         /* time LS started               */
       unsigned long   stop_time;          /* time LS stopped               */
       unsigned long   up_time;            /* total time LS active          */
       unsigned long   current_state_time; /* time in current state         */
       unsigned char   deact_cause;        /* deactivation cause            */
       unsigned char   hpr_support;        /* TG HPR support                */
       unsigned char   anr_label[2];       /* local ANR label               */
       unsigned char   hpr_link_lvl_error; /* HPR link-level error          */
       unsigned char   auto_act;           /* auto activate                 */
       unsigned char   ls_role;            /* link station role             */
       unsigned char   reserva;            /* reserved                      */
       unsigned char   node_id[4];         /* determined node id            */
       unsigned short  active_isr_count;   /* currently active ISR sessions */
       unsigned short  active_lu_sess_count;
                                           /* active LU-LU session count    */
       unsigned short  active_sscp_sess_count;
                                           /* active SSCP session count     */
       ANR_LABEL       reverse_anr_label;  /* reverse ANR label             */
       LINK STATION    local_address;      /* local LS address              */
       unsigned short  max_send_btu_size;  /* negotiated max BTU length     */
       unsigned char   brnn_link_type;     /* branch link type              */
       unsigned char   adj_cp_is_brnn;     /* adjacent CP is a BrNN         */
       unsigned char   mltg_member;        /* Reserved                      */
       unsigned char   tg_sharing;         /* Reserved                      */
       unsigned char   reservb[6];         /* reserved                      */
} LS_DET_DATA;

typedef struct anr_label
{
       unsigned short  length;           /* ANR label length                */
       unsigned short  reserv;           /* reserved                        */
       unsigned char   label[MAX_ANR_LABEL_SIZE];
                                         /* ANR label                       */
} ANR_LABEL;

typedef struct ls_def_data
{
       unsigned char   description[RD_LEN];
                                           /* resource description          */
       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 flag */
        unsigned char   dspu_services;      /* Services provided for          */
                                      /* downstream PU                 */
        unsigned char   dspu_name[8];       /* Downstream PU name (reserved   */
                                      /* if dspu_services is set to    */
                                      /* AP_NONE or AP_DLUR)           */
        unsigned char   dlus_name[17];      /* DLUS name if dspu_services     */
                                      /* is set to AP_DLUR             */
        unsigned char   bkup_dlus_name[17]; /* Backup DLUS name if            */
                                      /* dspu_services is set          */
                                      /* to AP_DLUR                    */
        unsigned char   hpr_supported;      /* does the link support HPR?     */
        unsigned char   hpr_link_lvl_error; /* does the link support HPR      */
                                      /* link-level error recovery?    */
        unsigned short  link_deact_timer;   /* HPR link deactivation timer    */
        unsigned char   reserv1;            /* reserved                       */
        unsigned char   default_nn_server;  /* Use as 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             */
        unsigned short  target_pacing_count;
                                      /* target pacing count           */
        unsigned short  max_send_btu_size;  /* max send BTU size              */
        unsigned char   ls_role;            /* link station role to use       */
                                      /* on this link                  */
        unsigned char   max_ifrm_rcvd;      /* max number of I-frames rcvd    */
        unsigned short  dlus_retry_timeout; /* DLUS retry timeout             */
        unsigned short  dlus_retry_limit;   /* DLUS retry limit               */
        unsigned char   conventional_lu_compression;
                                      /* Data compression requested for */
                                      /* conventional LU sessions      */
        unsigned char   conventional_lu_cryptography;
                                      /* Cryptography required for     */
                                      /* conventional LU sessions      */
        unsigned char   reserv3;            /* reserved                       */
        unsigned char   retry_flags;        /* conditions for automatic       */
                                      /* retries                       */
        unsigned short  max_activation_attempts;
                                      /* how many automatic retries:   */
        unsigned short  activation_delay_timer;
                                      /* delay between automatic       */
                                      /* retries                       */
        unsigned char   branch_link_type;   /* branch link type               */
        unsigned char   adj_brnn_cp_support;/* adjacent BrNN CP support       */
        unsigned char   reserv4[20];        /* reserved                       */
        unsigned short  link_spec_data_len; /* length of link specific data   */
} LS_DEF_DATA;

typedef struct link_address
{
        unsigned short  length;             /* length                         */
        unsigned short  reserve1;           /* reserved                       */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                      /* address                       */
} LINK_ADDRESS;
```

```
typedef struct link_spec_data
{
      unsigned char  link_data[SIZEOF_LINK_SPEC_DATA];

}  LINK_SPEC_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;        /* Modem class                   */
        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
{
        unsigned long   in_xid_bytes;       /* number of XID bytes received  */
        unsigned long   in_msg_bytes;       /* num message bytes received    */
        unsigned long   in_xid_frames;      /* num XID frames received       */
        unsigned long   in_msg_frames;      /* num message frames received   */
        unsigned long   out_xid_bytes;      /* num XID bytes sent            */
        unsigned long   out_msg_bytes;      /* num message bytes sent        */
        unsigned long   out_xid_frames;     /* num XID frames sent           */
        unsigned long   out_msg_frames;     /* num message frames sent       */
        unsigned long   in_invalid_sna_frames;
                                            /* num invalid frames received   */
        unsigned long   in_session_control_frames;
                                            /* num control frames received   */
        unsigned long   out_session_control_frames;
                                            /* num control frames sent       */
        unsigned long   echo_rsps;          /* response from adj LS count    */
        unsigned long   current_delay;      /* time taken for last test sig  */
        unsigned long   max_delay;          /* max delay by test signal      */
        unsigned long   min_delay;          /* min delay by test signal      */
        unsigned long   max_delay_time;     /* time since longest delay      */
        unsigned long   good_xids;          /* successful XID on LS count    */
        unsigned long   bad_xids;           /* unsuccessful XID on LS count  */
} LS_STATS;
```

VCB Structure

Format 0 (back-level)

```
typedef struct ls_det_data
{
        unsigned short  act_sess_count;     /* curr 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;            /* is LS is dynamic ?            */
        unsigned char   migration;          /* supports migration partners   */
        unsigned char   tg_num;             /* TG number                     */
        LS_STATS        ls_stats;           /* link station statistics       */
        unsigned long   start_time;         /* time LS started               */
        unsigned long   stop_time;          /* time LS stopped               */
        unsigned long   up_time;            /* total time LS active          */
        unsigned long   current_state_time; /* time in current state         */
        unsigned char   deact_cause;        /* deactivation cause            */
        unsigned char   hpr_support;        /* TG HPR support               */
        unsigned char   anr_label[2];       /* local ANR label              */
        unsigned char   hpr_link_lvl_error; /* HPR link-level error         */
        unsigned char   auto_act;           /* auto activate                */
        unsigned char   ls_role;            /* link station role            */
        unsigned char   reserva;            /* reserved                     */
        unsigned char   node_id[4];         /* determined node id           */
        unsigned short  active_isr_count;   /* currently active ISR sessions */
        unsigned char   reservb[30];        /* reserved                     */
} LS_DET_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_LS

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to one to specify the format 1 version of the VCB listed above. If this is set to 0, the Program returns the format 0 LS_DET_DATA structure.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **ls_name** specified (see the following parameter, **ls_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

ls_name
Link station name. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

port_name
Port name filter. This should be set to all zeros or an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set then only link stations belonging to this port are returned. This field is ignored if it is set to all zeros.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

ls_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

ls_summary.ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

ls_summary.description
Resource description (as specified on DEFINE_LS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

ls_summary.dlc_type
Type of DLC. The Program supports the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

Additional DLC types can be defined by specifying the new type on the DEFINE_DLC verb. See “DEFINE_DLC” on page 46, for more information.

ls_summary.state
State of this link station. This field is set to one of the following values:

AP_NOT_ACTIVE
AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE

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, 17-byte, adjacent CP name determined during link activation. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This will be null if the LS is inactive.

If **ls_summary.adj_cp_type** is not one of AP_NETWORK_NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE, then this field is reserved.

ls_summary.det_adj_cp_type
Type of the adjacent node determined during link activation. It is one of the following values:

AP_END_NODE
AP_NETWORK_NODE
AP_LEARN_NODE
AP_VRN

This will be AP_LEARN_NODE if the LS is inactive.

If **ls_summary.adj_cp_type** is not one of AP_NETWORK_NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE, then this field is reserved.

ls_summary.port_name
Name of port associated with this link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

ls_summary.adj_cp_name
Fully qualified, 17-byte, adjacent control point name composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This will be null for an implicit link.

ls_summary.adj_cp_type
Type of the adjacent node. It is one of the following values:

AP_END_NODE
AP_NETWORK_NODE
AP_APPN_NODE
AP_BACK_LEVEL_LEN__NODE
AP_HOST_XID3

AP_HOST_XID0
AP_DSPU_XID
AP_DSPU_NOXID

ls_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

ls_detail.ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

ls_detail.det_data.act_sess_count
Total number of active sessions (both endpoint and intermediate) using the link.

ls_detail.det_data.dlc_type
Type of DLC. The Program supports the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

Additional DLC types can be defined by specifying the new type on the DEFINE_DLC verb. See "DEFINE_DLC" on page 46, for more information.

ls_detail.det_data.state
State of this link station. This field is set to one of the following values:

AP_NOT_ACTIVE
AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE

ls_detail.det_data.sub_state
This field provides more detailed information about the state of this link station. This field is set to one of the following values:

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, 17-byte, adjacent control point name determined during link activation. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If **ls_summary.adj_cp_type** is not one of AP_NETWORK_NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE, then this field is reserved.

ls_detail.det_data.det_adj_cp_type
Type of the adjacent node determined during link activation. It is one of the following values:

AP_END_NODE
AP_NETWORK_NODE
AP_LEARN_NODE
AP_VRN

If **ls_summary.adj_cp_type** is not one of AP_NETWORK_NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE, then this field is reserved.

ls_detail.det_data.dlc_name
Name of the DLC. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

ls_detail.det_data.dynamic
Specifies whether the link was defined explicitly (by a DEFINE_LS command), or implicitly or dynamically (either in response to a connection request from the adjacent node, or to connect dynamically to another node across a connection network). This can be AP_YES or AP_NO.

ls_detail.det_data.migration
Specifies whether the adjacent node is a migration level node (such as a low entry networking (LEN) node), or a full APPN network node or end node (AP_YES, AP_NO, or AP_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 SNA incorrect 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.echo_rsps
Number of echo responses received from the adjacent node. Echo requests are sent periodically to gauge the propagation delay to the adjacent node.

ls_detail.det_data.ls_stats.current_delay
Time (in milliseconds) that it took for the last test signal to be sent and returned from this link station to the adjacent link station.

ls_detail.det_data.ls_stats.max_delay
Longest time taken (in milliseconds) for a test signal to be sent and returned from this link station to the adjacent link station.

ls_detail.det_data.ls_stats.min_delay
Shortest time taken (in milliseconds) for a test signal to be sent and returned from this link station to the adjacent link station.

ls_detail.det_data.ls_stats.max_delay_time
Time since system startup (in hundredths of a second) when the longest delay occurred.

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, 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
The total time (in hundredths of a second) that this link station has been active since system startup.

ls_detail.det_data.current_state_time
The total time (in hundredths of a second) that this link station has been in the current state.

ls_detail.det_data.deact_cause
The cause of the last deactivation of the link station. The field is set to one of the following values:

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
The level of high-performance routing (HPR) supported on this TG (AP_NONE, AP_BASE or AP_RTP), taking account of the capabilities of the local and adjacent nodes.

ls_detail.det_data.anr_label
The HPR automatic network routing (ANR) label allocated to the local link.

ls_detail.det_data.hpr_link_lvl_error
Specifies whether link-level error recovery is being used for HPR traffic on the link.

ls_detail.def_data.auto_act
Specifies whether the link currently allows remote activation or activation on demand. The following values are returned (and may be ORed together:

AP_AUTO_ACT
The link can be activated on demand by the local node.

AP_REMOTE_ACT
The link can be activated by the remote node.

ls_detail.det_data.ls_role
The link station role for this link station. This is initially set to the link station role defined for the link station. If the defined role is negotiable, this value changes to the negotiated role (primary or secondary) during the XID exchange, and reverts back to negotiable when the link is deactivated.

AP_LS_NEG
The link station role is negotiable.

AP_LS_PRI
The link station role is primary.

AP_LS_SEC
The link station role is secondary.

ls_detail.det_data.node_id
Node ID received from adjacent node during XID exchange. This 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 the link.

ls_detail.det_data.active_sscp_sess_count
The count of active LU-SSCP and PU-SSCP sessions using the link.

ls_detail.det_data.reverse_anr_label.length
: The length of the reverse Automatic Network Routing (ANR) label for the link station. If the link does not support HPR, or the label is not known, this field is zeroed.

ls_detail.det_data.reverse_anr_label.label
: The reverse Automatic Network Routing (ANR) label for the link station. If the link does not support HPR, or the label is not known, this field is zeroed.

ls_detail.det_data.local_address
: The local address of this link station.

ls_detail.det_data.max_send_btu_size
: The maximum BTU size that can be sent on this link, as determined by negotiation with the adjacent node. If link activation has not yet been attempted, zero is returned.

ls_detail.det_data.brnn_link_type
: BrNN only. This branch link type. It is one of the following:

  AP_UPLINK
  : This link is an uplink.

  AP_DOWNLINK
  : The link is a downlink.

  AP_OTHERLINK
  : This link is an otherlink.

  AP_UNKNOWN_LINK_TYPE
  : This link is an otherlink.

  AP_BRNN_NOT_SUPPORTED
  : This link supports PU 2.0 traffic only.

  Other node types: This field is not meaningful and is always set to AP_BRNN_NOT_SUPPORTED.

ls_detail.det_data.adj_cp_is_brnn
: All node types: Specifies whether the adjacent node is a BrNN.

  AP_UNKNOWN
  : It is not known whether the adjacent node is a BrNN.

  AP_NO
  : The adjacent node is not a BrNN.

  AP_YES
  : The adjacent node is BrNN.

ls_detail.def_data.description
: Resource description (as specified on DEFINE_LS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

ls_detail.def_data.port_name
: Name of port associated with this link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant. 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 17-byte adjacent control point name, which is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and

is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This is defined if **back_lvl_len_end_node** is not set to AP_NO, or if the port associated with the link station is defined to be switched.

ls_detail.def_data.adj_cp_type
Adjacent node type.

AP_NETWORK_NODE
Specifies that the node is an APPN network node.

AP_END_NODE
Specifies that the node is an APPN end node or an up-level LEN node.

AP_APPN_NODE
Specifies that the node is an APPN network node, an APPN end node, or an up-level LEN node. The node type will be learned during XID exchange.

AP_BACK_LEVEL_LEN_NODE
Specifies that the node is a back-level LEN node.

AP_HOST_XID3
Specifies that the node is a host and that the Node Operator Facility responds to a polling XID from the node with a format 3 XID.

AP_HOST_XID0
Specifies that the node is a host and that the Node Operator Facility responds to a polling XID from the node with a format 0 XID.

AP_DSPU_XID
Specifies that the node is a downstream PU and that the Node Operator Facility includes XID exchange in link activation.

AP_DSPU_NOXID
Specifies that the node is a downstream PU and that the Node Operator Facility does not include XID exchange in link activation.

Note: A link station to a VRN is always dynamic and is therefore not defined.

ls_detail.def_data.dest_address.length
Length of destination link station's address on adjacent node.

ls_detail.def_data.dest_address.address
Link station's destination address on adjacent node.

ls_detail.def_data.auto_act_supp
Specifies whether the link will be activated automatically after it has been started by a START_LS verb, and stopped by a STOP_LS. (AP_YES or AP_NO).

ls_detail.def_data.tg_number
Preassigned TG number (in the range one to 20). This number is used to represent the link when the link is activated. Zero indicates that the TG number is not preassigned but 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. This is set to one of the following values:

AP_NO
The link is not a limited resource and will not be deactivated automatically.

AP_YES or 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
AP_YES requests the host to initiate sessions between the SSCP and the local control point and dependent LUs. AP_NO requests no sessions with the SSCP on this link.

ls_detail.def_data.pu_name
Name of the local PU that is going to use this link if **solicit_sscp_sessions** is set to AP_YES. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If **solicit_sscp_sessions** is set to AP_NO, this field is reserved.

ls_detail.def_data.disable_remote_act
Specifies whether remote activation of this link is supported (AP_YES or AP_NO).

ls_detail.def_data.dspu_services
Specifies the services that the local node provides to the downstream PU across this link if **solicit_sscp_sessions** is set to AP_NO. This is set to one of the following:

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.

If **solicit_sscp_sessions** is set to AP_YES, this field is reserved.

ls_detail.def_data.dspu_name
Name of the downstream PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This is only valid if **solicit_sscp_sessions** is set to AP_NO.

ls_detail.def_data.dlus_name
Name of DLUS node which DLUR solicits SSCP services from when the link to the downstream node is activated. This is either set to all zeros or a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, then the global default DLUS (if defined by the DEFINE_DLUR_DEFAULTS verb) is solicited when the link is activated. If the **dlus_name** is set to zeros and there is no global default DLUS, then DLUR will not initiate SSCP contact when the link is activated. This field is reserved if **dspu_services** is not set to AP_DLUR.

ls_detail.def_data.bkup_dlus_name
Name of DLUS node which serves as the backup for the downstream PU. This is either set to all zeros or to a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the field is set to all zeros, then the global backup default DLUS (if defined by the DEFINE_DLUR_DEFAULTS verb) is used as the backup for this PU. This field is reserved if **dspu_services** is not set to AP_DLUR.

ls_detail.def_data.hpr_supported
Specifies whether HPR is supported on this link (AP_YES or AP_NO).

ls_detail.def_data.hpr_link_lvl_error
Specifies whether the HPR link-level error recovery tower is supported on this link (AP_YES or AP_NO). Note that the parameter is reserved if **hpr_supported** is set to AP_NO.

ls_detail.def_data.link_deact_timer
Limited resource link deactivation timer (in seconds).

If **limited_resource** is set to AP_YES or AP_NO_SESSIONS, a link is automatically deactivated if no data traverses the link for the duration of this timer, and no sessions are using the link.

If **limited_resource** is set to AP_INACTIVITY then a link is automatically deactivated if no data traverses the link for the duration of this timer.

ls_detail.def_data.default_nn_server
Specifies whether a link can be automatically activated by an end node to support CP-CP sessions to a network node server. (AP_YES or AP_NO). The link must be defined to support CP-CP sessions for this field to take effect.

ls_detail.def_data.ls_attributes
Specifies further information about the adjacent node.

ls_detail.def_data.ls_attributes[0]
Host type.

AP_SNA
Standard SNA host.

AP_FNA
FNA (VTAM-F) host.

AP_HNA
HNA host.

ls_detail.def_data.ls_attributes[1]
This is a bit field. It may take the value AP_NO, or any of the following values bitwise ORed together

AP_SUPPRESS_CP_NAME
Network Name CV suppression option for a link to a back-level LEN node. If this bit is set, no Network Name CV is included in XID exchanges with the adjacent node. (This bit is ignored unless **adj_cp_type** is set to AP_BACK_LEVEL_LEN_NODE or AP_HOST_XID3.)

AP_REACTIVATE_ON_FAILURE
If the link is active and then fails, Personal Communications or

Communications Server will attempt to reactivate the link. If the reactivation attempt fails, the link will remain inactive.

AP_USE_PU_NAME_IN_XID_CVS
If the adjacent node is defined to be a host or **solicit_sscp_sessions** is set to AP_YES on a link to an APPN node, and the AP_SUPPRESS_CP_NAME bit is not set, then the fully-qualified CP name in Network Name CVs sent on Format 3 XIDs is replaced by the name supplied in **def_data.pu_name**, fully-qualified with the network ID of the CP.

ls_detail.def_data.adj_node_id
Defined node ID of adjacent node.

ls_detail.def_data.local_node_id
Node ID sent in XIDs on this link station. This is a 4-byte hexadecimal string. If this field is set to zero, the **node_id** is used in XID exchanges. If this field is nonzero, it replaces the value for XID exchanges on this LS.

ls_detail.def_data.cp_cp_sess_support
Specifies whether CP-CP sessions are supported (AP_YES or AP_NO).

ls_detail_def_data.use_default_tg_chars
Specifies whether the TG characteristics supplied on the DEFINE_LS were discarded in favor of the default characteristics supplied on the DEFINE_PORT (AP_YES or AP_NO). This field does not apply to implicit links.

ls_detail.def_data.tg_chars
TG characteristics (See "DEFINE_CN" on page 31).

ls_detail.def_data.target_pacing_count
Numeric value between 1 and 32 767 inclusive indicating the desired pacing window size for BINDs on this TG. The number is only significant when fixed bind pacing is being performed. Note that Personal Communications or Communications Server does not currently use this value.

ls_detail.def_data.max_send_btu_size
Maximum BTU size that can be sent.

ls_detail.def_data.ls_role
The link station role that this link station should assume. This can be any one of AP_LS_NEG, AP_LS_PRI or AP_LS_SEC to select a role of negotiable, primary or secondary. The field can also be set to AP_USE_PORT_DEFAULTS to select the value configured on the DEFINE_PORT verb.

ls_detail.def_data.max_ifrm_rcvd
The maximum number of I-Frames that can be received by the XID sender before acknowledgment. Set to zero if the default value from DEFINE_PORT should be used.

ls_detail.def_data.dlus_retry_timeout
Interval in seconds between second and subsequent attempts to contact DLUS specified in the **Is_detail.def_data.dlus_name** and **Is_detail.def_data.bkup_dlus_name** fields. The interval between the initial attempt and the first retry is always one second. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. This field is ignored if **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 a DLUS specified in the **Is_detail.def_data.dlus_name** and **Is_detail.def_data.bkup_dlus_name** fields. If zero is specified, the default value configured through DEFINE_DLUR_DEFAULTS is used. If X'FFFF' is specified, the Program retries indefinitely. This field is ignored if **def_data.dspu_services** is not set to AP_DLUR.

ls_detail.def_data.link_spec_data_len
Unpadded length, in bytes, of data passed unchanged to link station component during initialization. The data is concatenated to the LS_DETAIL structure. This data will be padded to end on a 4-byte boundary.

ls_detail.def_data.conventional_lu_compression
Specifies whether data compression is requested for sessions on this link. Note that this field is only valid for links carrying LU 0 to 3 traffic.

AP_NO
The local node should not compress or decompress conventional LU data flowing over this link.

AP_YES
Data compression should be enabled for conventional LU sessions on this link if the host requests compression.

ls_detail.def_data.conventional_lu_cryptography
Specifies whether session level encryption is required for conventional LU sessions. This field only applies for links carrying conventional LU traffic.

AP_NONE
Session level encryption is not performed by the Program.

AP_MANDATORY
Mandatory session level encryption is performed by the Program if an import key is available to the LU. Otherwise, it must be performed by the application that uses the LU (if this is PU Concentration, it is performed by a downstream LU).

AP_OPTIONAL
This value allows the cryptography used to be driven by the host application on a per session basis. If the host request cryptography for a session dependent on this PU, then the behaviour of the Program is as for AP_MANDATORY. If the host does not request cryptography, then the behaviour is the same as AP_NONE.

ls_detail.def_data.retry_flags
This field specifies the conditions under which activation of this link station is subject to automatic retry. It is a bit field, and may take any of the following values bit-wise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, the Program will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, the Program attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_DELAY_INHERIT_RETRY
In addition to the retry conditions specified by flags in this field, those specified in the **retry_flags** field of the underlying port definition will also be used.

ls_detail.def_data.max_activation_attempts
This field has no effect unless at least one flag is set in **retry_flags**.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means 'no limit'. The value AP_USE_DEFAULTS results in the use of **max_activiation_attempts** supplied on DEFINE_PORT.

ls_detail.def_data.activation_delay_timer
This field has no effect unless at least one flag is set in **retry_flags**.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **def_data.retry_flags**.

The value AP_USE_DEFAULTS results in the use of **activiation_delay_timer** supplied on DEFINE_PORT.

If zero is specified, the Program uses a default timer duration of thirty seconds.

ls_detail.def_data.branch_link_type
BrNN only. Specifies whether a link is an uplink or a downlink. This field only applies if the field **def_data.adj_cp_type** is set to AP_NETWORK, NODE, AP_END_NODE, AP_APPN_NODE, or AP_BACK_LEVEL_LEN_NODE.

AP_UPLINK
This link is an uplink.

AP_DOWNLINK
The link is a downlink.

If the field **adj_cp_type** is set to AP_NETWORK_NODE, then this field must be set to AP_UPLINK.

Other node types: This field is ignored.

ls_detail.det_data.adj_brnn_cp_support
BrNN only. Specifies whether the adjacent CP is allowed to be, required to be, or prohibited from being an NN(BrNN), for example, a BrNN showing

its NN face. This field only applies if the field **adj_cp_type** is set to AP_NETWORK_ NODE or AP_APPN_NODE, (and the node type learned during XID exchange is network node).

AP_BRNN_ALLOWED
The adjacent CP is allowed (but not required) to be an NN(BrNN).

AP_BRNN_REQUIRED
The adjacent CP is not allowed to be an NN(BrNN).

AP_BRNN_PROHIBITED
The adjacent CP is not allowed to be an NN(BrNN).

If the field **adj_cp_type** is set to AP_NETWORK_NODE and the field **auto_act_supp** is set to AP_YES, then this field must be set to AP_BRNN_REQUIRED or AP_BRNN_PROHIBITED.

Other node types: This field is ignored.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LINK_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_LS_EXCEPTION

QUERY_LS_EXCEPTION 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) and defined data (the data supplied by the application on DEFINE_LS).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LS, or to obtain the list information in several chunks, the **ls_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **ls_name**. Ordering is according to name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The list of link stations returned can be filtered by the name of the port that they belong to. In this case, the **port_name** field should be set (otherwise this field should be set to all zeros).

VCB Structure

```
typedef struct query_ls_exception
{
        unsigned short  opcode;            /* verb operation code           */
        unsigned char   reserv2;           /* reserved                      */
        unsigned char   format;            /* format                        */
        unsigned short  primary_rc;        /* Primary return code           */
        unsigned long   secondary_rc;      /* Secondary return code         */
        unsigned char   *buf_ptr;          /* pointer to buffer             */
        unsigned long   buf_size;          /* buffer size                   */
        unsigned long   total_buf_size;    /* total buffer size required    */
        unsigned short  num_entries;       /* number of entries             */
        unsigned short  total_num_entries; /* total number of entries       */
        unsigned char   list_options;      /* listing options               */
        unsigned char   reserv3;           /* reserved                      */
        unsigned long   exception_index;   /* index of LS exception entry   */
        unsigned char   ls_name;           /* name of link station          */
} QUERY_LS_EXCEPTION;
typedef struct LS_EXCEPTION
{
        unsigned short  overlay_size;      /* size of this entry            */
        unsigned long   exception_indes;   /* index of this entry           */
        unsigned_DATE_TIME
                        time;              /* date and time                 */
        unsigned char   ls_name[8];        /* link station name             */
        unsigned char   adj_cp_name[17];
                                           /* adjacent CP name              */
        unsigned char   adj_node_id[4];
                                           /* adjacent node id              */
        unsigned short  tg_number;         /* TG number                     */
        unsigned long   general_sense;     /* general sense data            */
        unsigned char   retry;             /* wil retry request             */
        unsigned long   end_sense;         /* termination sense data        */
        unsigned long   xid_local_sense;   /* XID local sense data          */
        unsigned long   xid_remote_sense;  /* XID remote sense data         */
        unsigned short  xid_error_byte;     /* offset of byte in error        */
        unsigned short  xid_error_bit;      /* offset of bit in error         */
        unsigned char   dlc_type;           /* DLC type                       */
        LINK_ADDRESS    local_addr;         /* local address                  */
        LINK_ADDRESS    destination_addr;   /* destination address            */
        unsigned char   reserved[20];       /* reserved                       */
} LS_EXCEPTION;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_LS_EXCEPTION

format
Identifies the format of the VCB. Set this field to one to specify the format 1 version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

The **index** specified in the following parameter represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignore and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned lists starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

exception_index
Index of the LS exception entry. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

ls_name
Name of the link station that returned entries relate. This is an 8–byte string in a locally displayable character set. All 8 bytes are significant. If this field is set to null, then entries that relate to any or all links stations are returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

ls_exception.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

ls_exception.exception_index
The index assigned for this LS exception entry. The value of the index begins at zero and increases incrementally up to a maximum value of 2**31–1 (2,147,483,647) before wrapping.

ls_exception.time
Time and date that the LS exception entry was generated.

ls_exception.ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

ls_exception.adj_cp_name
Fully qualified, 17-byte, adjacent CP name. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) The value of this field is determined as follows:

If an adjacent CP name was received on XID, it is returned.

If an adjacent CP name was received on XID, but a locally-defined value is available, it is returned.

Otherwise, null is returned.

ls_exception.node_id
Node ID received from adjacent node during XID exchange (or null if none is received). This is a 4–byte hexadecimal string.

ls_exception.tg_number
Number associated with the TG to this link station. Range 0 through 256. A value of 256 indicates that the TG number was unknown at the time of the failure.

ls_exception.general_sense
The error sense data associated with the start sequence of activation of a link up to the beginning of the XID sequence. This is generated by the node.

ls_exception.retry
Indicates whether the node will retry the start request to activate the link.

AP_NO
The node will not retry the start request.

AP_YES
The node will retry the start request.

ls_exception.end_sense
Sense data associated with the termination of the activation attempt. This is generated by the DLC layer.

ls_exception.xid_local_sense
Locally generated sense data sent in XID.

ls_exception.xid.remote_sense
Remotely generated sense data received in XID.

ls_exception.xid_error_byte
Offset of error bit in error byte in XID (range 0 through 65535). The value 65535 indicates that this field has no meaning.

ls_exception.xid_error_bit
Offset of error bit in error byte in XID (range 0 through 7). The value 8 indicates that this field has no meaning.

ls_exception.dlc_type
Type of DLC. The Program supports the following types:

AP_SDLC
AP_X25
AP_TR

Additional DLC types can be defined by specifying the new type on the DEFINE_DLC verb. See "DEFINE_DLC" on page 46, for more information.

ls_exception.local_addr.length
The length of local link station's address.

ls_exception.local_addr.address
The local link station's address.

ls_exception.destination_addr.length
The length of destination link station's address on adjacent node.

ls_exception.destination_addr.address
Destination link station's address on adjacent node.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_EXCEPTION_INDEX

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

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) and defined data (the data supplied by the application on DEFINE_LU_0_TO_3).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific local LU, or to obtain the list information in several chunks, the **lu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored.

Only certain parameters are supported on SNA API clients. See the note pad icon throughout this chapter for specific details.

This icon represents important information that can affect the operation of Communications Server and Personal Communications.

VCB Structure

```
typedef struct query_lu_0_to_3
{
        unsigned short  opcode;           /* verb operation code           */
        unsigned char   attributes;       /* Verb attributes               */
        unsigned char   reserv2;          /* reserved                      */
        unsigned char   format;           /* format                        */
        unsigned short  primary_rc;       /* primary return code           */
        unsigned long   secondary_rc;     /* secondary return code         */
        unsigned char   *buf_ptr;         /* pointer to buffer             */
        unsigned long   buf_size;         /* buffer size                   */
        unsigned long   total_buf_size;   /* total buffer size required    */
        unsigned short  num_entries;      /* number of entries             */
        unsigned short  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
{
        unsigned short  overlay_size;     /* size of this entry            */
        unsigned char   pu_name[8];       /* PU name                       */
        unsigned char   lu_name[8];       /* LU name                       */
        unsigned char   description[RD_LEN];
                                          /* resource description          */
        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
{
        unsigned short  overlay_size;     /* size of this 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;     /* LU-SSCP session statistics  */
        SESSION_STATS   plu_stats;         /* PLU-SLU session statistics  */
        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;          /* Application type            */
        unsigned char   sscp_id[6];        /* SSCP ID                     */
        unsigned char   bind_lu_type;      /* LU type issuing BIND        */
        unsigned char   reserva[12];       /* reserved                    */
} LU_0_TO_3_DET_DATA;
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; /* max send BTU size           */
        unsigned short  max_rcv_btu_size;  /* max rcv BTU size            */
        unsigned short  max_send_pac_win;  /* max send pacing win size    */
        unsigned short  cur_send_pac_win;  /* current send pacing win size */
        unsigned short  max_rcv_pac_win;   /* max receive pacing win 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 of FMD data frames sent */
        unsigned long   send_data_bytes;   /* number of data bytes sent   */
        unsigned long   rcv_data_frames;   /* num data frames received    */
        unsigned long   rcv_fmd_data_frames;
                                           /* num of FMD data frames recvd */
        unsigned long   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[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[RD_LEN];
                                           /* resource description        */
        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                    */
        unsigned char   sscp_id[6];
        unsigned char   timeout;           /* Timeout                     */
        unsigned char   app_spec_def_data[16];
                                           /* Application Specified Data  */
        unsigned char   model_name[7];     /* LU model                    */
        unsigned char   reserv3[17];         /* reserved                  */
} LU_0_TO_3_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_LU_0_TO_3

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

AP_SUMMARY
Returns summary information only.

AP_SUMMARY value is also supported for SNA API clients.

AP_DETAIL
Returns detailed information.

The **lu_name** specified (see the following parameter, **lu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list

AP_FIRST_IN_LIST value is also supported for SNA API clients.

.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

The list_options value is ignored for SNA API clients.

pu_name
PU name. Only LUs that use this PU will be returned. If a list of all LUs is required then this field should be set to all binary zeros. The list_options value is ignored for SNA API clients. The list_options value is ignored for SNA API clients.

The pu_name value is ignored for SNA API clients.

host_attachment
Filter for host attachment.

AP_NONE
Return information about all LUs.

AP_NONE is the only value supported for SNA API clients.

AP_DLUR_ATTACHED
Return information about all LUs that are supported by DLUR.

AP_DIRECT_ATTACHED
Return information about only those LUs that are directly attached to the host system.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

lu_0_to_3_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

lu_0_to_3_summary.pu_name
Name of local PU that this LU is using. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

The lu_0_to_3_summary.pu_name value is not returned on SNA API clients.

lu_0_to_3_summary.lu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

lu_0_to_3_summary.description
Resource description (as specified on DEFINE_LU_0_TO_3). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

The lu_0_to_3_summary.description value is not returned on SNA API clients.

lu_0_to_3_summary.nau_address
Network addressable unit address of the LU, which is in the range 1–255.

The lu_0_to_3_summary.nau_address value is not returned on SNA API clients.

lu_0_to_3_summary.lu_sscp_sess_active
Specifies whether the LU-SSCP session is active (AP_YES or AP_NO).

The lu_0_to_3_summary.lu_sscp_sess_active value is not returned on SNA API clients.

lu_0_to_3_summary.appl_conn_active
Specifies whether an application is using the LU (AP_YES or AP_NO).

The lu_0_to_3_summary.aapl_conn_active value is not returned on SNA API clients.

lu_0_to_3_summary.plu_sess_active
Specifies whether the PLU-SLU session is active (AP_YES or AP_NO).

The lu_0_to_3_summary.plu_sess_active value is not returned on SNA API clients.

lu_0_to_3_summary.host_attachment
LU host attachment type:

AP_DLUR_ATTACHED
LU is attached to host system using DLUR.

AP_DIRECT_ATTACHED
LU is directly attached to host system.

lu_0_to_3_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

lu_0_to_3_detail.lu_name
Name of the local LU that is being queried. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

lu_0_to_3_detail.det_data.lu_sscp_sess_active
Specifies whether the LU-SSCP session is active (AP_YES or AP_NO).

lu_0_to_3_detail.det_data.appl_conn_active
Specifies whether this LU is currently being used by an application (AP_YES or AP_NO).

lu_0_to_3_detail.det_data.plu_sess_active
Specifies whether the PLU-SLU session is active (AP_YES or AP_NO).

lu_0_to_3_detail.det_data.host_attachment
LU host attachment type:

AP_DLUR_ATTACHED
LU is attached to host system using DLUR.

AP_DIRECT_ATTACHED
LU is directly attached to host system.

lu_0_to_3_detail.det_data.lu_sscp_stats.rcv_ru_size
This field is always reserved.

lu_0_to_3_detail.det_data.lu_sscp_stats.send_ru_size
This field is always reserved.

lu_0_to_3_detail.det_data.lu_sscp_stats.max_send_btu_size
Maximum BTU size that can be sent.

lu_0_to_3_detail.det_data.lu_sscp_stats.max_rcv_btu_size
Maximum BTU size that can be received.

lu_0_to_3_detail.det_data.lu_sscp_stats.max_send_pac_win
This field will always be set to zero.

lu_0_to_3_detail.det_data.lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

lu_0_to_3_detail.det_data.lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

lu_0_to_3_detail.det_data.lu_sscp_stats.cur_rcv_pac_win
This field will always be set to zero.

lu_0_to_3_detail.det_data.lu_sscp_stats.send_data_frames
Number of normal flow data frames sent.

lu_0_to_3_detail.det_data.lu_sscp_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

lu_0_to_3_detail.det_data.lu_sscp_stats.send_data_bytes
Number of normal flow data bytes sent.

lu _0_to_3_detail.det_data.lu_sscp_stats.rcv_data_frames
Number of normal flow data frames received.

lu_0_to_3_detail.det_data.lu_sscp_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

lu_0_to_3_detail.det_data.lu_sscp_stats.rcv_data_bytes
Number of normal flow data bytes received.

lu_0_to_3_detail.det_data.lu_sscp_stats.sidh
Session ID high byte.

lu_0_to_3_detail.det_data.lu_sscp_stats.sidl
Session ID low byte.

lu_0_to_3_detail.det_data.lu_sscp_stats.odai
Origin Destination Address Indicator. When bringing up a session, the sender of the ACTLU sets this field 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_0_to_3_detail.det_data.lu_sscp_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 this session with the link over which the session flows.

lu_0_to_3_detail.det_data.lu_sscp_stats.pacing_type
Receive pacing type in use on the LU-SSCP session. This will be set to AP_NONE.

lu_0_to_3_detail.det_data.plu_stats.rcv_ru_size
Maximum receive RU size.

lu_0_to_3_detail.det_data.plu_stats.send_ru_size
Maximum send RU size.

lu_0_to_3_detail.det_data.plu_stats.max_send_btu_size
Maximum BTU size that can be sent.

lu_0_to_3_detail.det_data.plu_stats.max_rcv_btu_size
Maximum BTU size that can be received.

lu_0_to_3_detail.det_data.plu_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

lu_0_to_3_detail.det_data.plu_stats.cur_send_pac_win
Current size of the send pacing window on this session.

lu_0_to_3_detail.det_data.plu_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

lu_0_to_3_detail.det_data.plu_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

lu_0_to_3_detail.det_data.plu_stats.send_data_frames
Number of normal flow data frames sent.

lu_0_to_3_detail.det_data.plu_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

lu_0_to_3_detail.det_data.plu_stats.send_data_bytes
Number of normal flow data bytes sent.

lu_0_to_3_detail.det_data.plu_stats.rcv_data_frames
Number of normal flow data frames received.

lu_0_to_3_detail.det_data.plu_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

lu_0_to_3_detail.det_data.plu_stats.rcv_data_bytes
Number of normal flow data bytes received.

lu_0_to_3_detail.det_data.plu_stats.sidh
Session ID high byte.

lu_0_to_3_detail.det_data.plu_stats.sidl
Session ID low byte.

lu_0_to_3_detail.det_data.plu_stats.odai
Origin Destination Address 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.

lu_0_to_3_detail.det_data.plu_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.

lu_0_to_3_detail.det_data.plu_stats.pacing_type
Receive pacing type in use on the PLU-SSCP session. This can take the values AP_NONE or AP_PACING_FIXED.

lu_0_to_3_detail.det_data.plu_name
Primary LU name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. (If the PLU-SLU session is inactive this field is reserved).

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
Reserved.

lu_0_to_3_detail.det_data.app_type
Reserved.

lu_0_to_3_detail.det_data.sscp_id
This is 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_YES, then this field will be zeroed.

lu_0_to_3_detail.det_data.bind_lu_type
The LU type of the LU that issued the original BIND. If there is an active LU-LU session, then this can be one of the following:

AP_LU_TYPE_0
AP_LU_TYPE_1
AP_LU_TYPE_2
AP_LU_TYPE_3
AP_LU_TYPE_6 (for downstream dependent LU 6.2)

If there is no active LU—LU session, then this field takes the following value:

AP_LU_TYPE_UNKNOWN

lu_0_to_3_detail.def_data.description
Resource description (as specified on DEFINE_LU_0_TO_3). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

lu_0_to_3_detail.def_data.nau_address
Network addressable unit address of the LU, which is in the range 1–255.

lu_0_to_3_detail.def_data.pool_name
Name of pool to which this LU belongs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If the LU does not belong to a pool, this field is set to all binary zeros.

lu_0_to_3_detail.def_data.pu_name
Name of the PU (as specified on the DEFINE_LS verb) that this LU will use. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

lu_0_to_3_detail.def_data.priority
LU priority when sending to the host. This is set to one of the following values:

AP_NETWORK
AP_HIGH
AP_MEDIUM
AP_LOW

lu_0_to_3_detail.def_data.lu_model
Model type and number of the LU. This is set to one of the following values:

AP_3270_DISPLAY_MODEL_2
AP_3270_DISPLAY_MODEL_3
AP_3270_DISPLAY_MODEL_4
AP_3270_DISPLAY_MODEL_5
AP_RJE_WKSTN
AP_PRINTER
AP_SCS_PRINTER
AP_UNKNOWN

lu_0_to_3_detail.def_data.sscp_id
This field specifies the ID of the SSCP permitted to activate this LU. It is a 6–byte binary field. If the field is set to binary zeros, then the LU may be activated by any SSCP.

lu_0_to_3_detail.def_data.timeout
Timeout for 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 holds:

- 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.

lu_0_to_3_detail.def_data.app_spec_def_data
Application-specified data from DEFINE_LU_0_TO_3; the Program does not interpret this field, it is simply stored and returned on the QUERY_LU_0_TO_3 verb.

lu_0_to_3_detail.def_data.model_name
The value returned is the value specified in the format 1 DEFINE_LU_0_TO_3 verb, or hex zeros if the DEFINE verb was format 0.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_LU_POOL

QUERY_LU_POOL returns a list of pools and the LUs that belong to them.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific LU pool or to obtain the list information in several chunks, the **pool_name** and **lu_name** fields should be set. If the **lu_name** field is set to all zeros, then the information returned starts from the first LU in the specified pool. If the **list_options** field is set to AP_FIRST_IN_LIST, then both of these fields are ignored.

VCB Structure

```
typedef struct query_lu_pool
{
        unsigned short  opcode;              /* verb operation code        */
        unsigned char   attributes;          /* verb attributes            */
        unsigned char   format;              /* format                     */
        unsigned short  primary_rc;          /* primary return code        */
        unsigned long   secondary_rc;        /* secondary return code      */
        unsigned char   *buf_ptr;            /* pointer to buffer          */
        unsigned long   buf_size;            /* buffer size                */
        unsigned long   total_buf_size;      /* total buffer size required  */
        unsigned short  num_entries;         /* number of entries          */
        unsigned short  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
{
        unsigned short  overlay_size;        /* size of this entry         */
        unsigned char   pool_name[8];        /* pool name                  */
        unsigned char   description[RD_LEN]; /* resource description       */
        unsigned short  num_active_lus;      /* num of currently active LUs */
        unsigned char   num_avail_lus;       /* num of currently available  */
                                             /* LUs                        */
} LU_POOL_SUMMARY;
```

```
typedef struct lu_pool_detail
{
        unsigned short  overlay_size;        /* size of this entry         */
        unsigned char   pool_name[8];        /* pool name                  */
        unsigned char   description[RD_LEN]; /* resource description       */
        unsigned char   lu_name[8];          /* LU name                    */
        unsigned char   lu_sscp_sess_active; /* Is LU-SSCP session active   */
        unsigned char   appl_conn_active;    /* Is SSCP 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

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **pool_name** and **lu_name** specified (see the following parameters, **pool_name** and **lu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

pool_name
Name of LU pool. This name is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

lu_name
LU name. This name is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this is set to all binary zeros, the LUs belonging to the specified pool are listed from the beginning of the pool. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of directory entries returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

lu_pool_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

lu_pool_summary.pool_name
Name of LU pool to which the specified LU belongs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. (Note that if this field is specified on the request and the **lu_name** field is set to all binary zeros, then only LUs in the pool are returned.)

lu_pool_summary.description
LU pool description (as specified on DEFINE_LU_POOL).

lu_pool_summary.num_active_lus
The number of LUs in the specified pool that have active LU-SSCP sessions.

lu_pool_summary.num_avail_lus
The number of LUs in the specified pool available to satisfy an OPEN_LU_SSCP_SEC_REQ with **open_force** set to AP_YES. It includes all LUs whose PU is active or whose host link is automatically activated, and whose connection is free. This count is regardless of the LU **model_type**, **model_name**, and the DDDLU support of the PU. There might be less LUs available to satisfy an OPEN_LU_SSCP_SEC_REQ that specifies a particular value for **model_type**.

lu_pool_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

lu_pool_detail.pool_name
Name of LU pool to which the specified LU belongs. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. (Note that if this field is specified on the request and the **lu_name** field is set to all binary zeros, then only LUs in the pool are returned.)

lu_pool_detail.description
LU description (as specified on DEFINE_LU_0_TO_3).

lu_pool_detail.lu_name
LU name of LU belonging to the pool. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this name is set to all zeros then, this indicates that the specified pool is empty.

lu_pool_detail.lu_sscp_sess_active
Specifies whether the LU-SSCP session is active (AP_YES or AP_NO).

lu_pool_detail.appl_conn_active
Specifies whether the LU session is currently being used by an application (AP_YES or AP_NO).

lu_pool_detail.plu_sess_active
Specifies whether the PLU-SLU session is active (AP_YES or AP_NO).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LIST_OPTION

AP_INVALID_POOL_NAME
AP_INVALID_LU_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_MDS_APPLICATION

QUERY_MDS_APPLICATION returns a list of applications that have registered for MDS level messages.

Applications register by issuing the REGISTER_MS_APPLICATION verb described in Chapter 15, “Management Services Verbs”, on page 611.

To obtain information about a specific application, or to obtain the list information in several chunks, the **application** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

VCB Structure

```
typedef struct query_mds_application
{
        unsigned short  opcode;            /* verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* primary return code       */
        unsigned long   secondary_rc;      /* secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  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
{
        unsigned short  overlay_size;      /* size of this entry        */
        unsigned char   application[8];    /* application name          */
        unsigned short  max_rcv_size;      /* max data size application  */
                                           /* can receive               */
        unsigned char   reserva[20];       /* reserved                  */
} MDS_APPLICATION_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_MDS_APPLICATION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **application** specified (see the following parameter, **application**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

application
Application name. The name is an 8-byte alphanumeric type-A EBCDIC character string. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

mds_application_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mds_application_data.application
Name of registered application. The name is an 8-byte alphanumeric type-A EBCDIC character string.

mds_application_data.max_rcv_size
The maximum number of bytes that the application can receive in one chunk (this is specified when an application registers with MDS). For more information about MDS-level application registration refer to Chapter 15, “Management Services Verbs”, on page 611.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_APPLICATION_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_MDS_STATISTICS

QUERY_MDS_STATISTICS returns management services statistics. This verb can be used to gauge the level of MDS routing traffic.

VCB Structure

```
typedef struct query_mds_statistics
{
        unsigned short  opcode;             /* verb operation code        */
        unsigned char   reserv2;            /* reserved                   */
        unsigned char   format;             /* format                     */
        unsigned short  primary_rc;         /* primary return code        */
        unsigned long   secondary_rc;       /* secondary return code      */
        unsigned long   alerts_sent;        /* number of alert sends      */
        unsigned long   alert_errors_rcvd;  /* error messages received    */
                                            /* for alert sends            */
        unsigned long   uncorrelated_alert_errors;
                                            /* uncorrelated alert         */
                                            /* errors received            */
        unsigned long   mds_mus_rcvd_local; /* number of MDS_MUs received  */
                                            /* from local applications    */
        unsigned long   mds_mus_rcvd_remote;
                                            /* number of MDS_MUs received  */
                                            /* from remote applications   */
        unsigned long   mds_mus_delivered_local;
                                            /* num of MDS_MUs delivered   */
                                            /* to local applications      */
        unsigned long   mds_mus_delivered_remote;
                                            /* num of MDS_MUs             */
                                            /* delivered to remote appls  */
        unsigned long   parse_errors;       /* number of MDS_MUs received  */
                                            /* with parse errors          */
        unsigned long   failed_deliveries;  /* number of MDS_MUs where    */
                                            /* delivery failed            */
        unsigned long   ds_searches_performed;
                                            /* number of DS searches done  */
        unsigned long   unverified_errors;  /* number of unverified errors */
        unsigned char   reserva[20];        /* reserved                   */
} QUERY_MDS_STATISTICS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_MDS_STATISTICS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

If the verb executes successfully, the Program 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 can be held until the problem determination focal point is restored.

Note: The two counts, **alert_errors_rcvd** and **uncorrelated_errors_rcvd** are maintained such that the size of the send alert queue can be tuned. An increasing **uncorrelated_errors_rcvd** over time 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 that contained header format errors.

failed_deliveries
Number of MDS_MUs this node failed to deliver.

ds_searches_performed
Reserved.

unverified_errors
Reserved.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_MODE

QUERY_MODE returns information about modes that have been used by a local LU with a particular partner LU. The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific mode, or to obtain the list information in several chunks, the **mode_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. Note that the **lu_name** (or **lu_alias**) and **plu_alias** (or **fqplu_name**) fields must always be set. The **lu_name**, if nonzero, will be used in preference to the **lu_alias**. See “Querying the Node” on page 10, for background on how the list formats are used.

The list only includes information for the local LU specified by the **lu_name** (or **lu_alias**). This list is ordered by the **fqplu_name** followed by the **mode_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If **plu_alias** is set to all zeros, the **fqplu_name** value will be used, otherwise the **plu_alias** is always used and the **fqplu_name** is ignored.

The list of modes returned can be filtered according to whether they currently have any active sessions. If filtering is desired, the **active_sessions** field should be set to AP_YES (otherwise this field should be set to AP_NO). This verb returns information that is determined once the mode begins to be used by a local LU with a partner LU. The QUERY_MODE_DEFINITION verb returns definition information only.

VCB Structure

```
typedef struct query_mode
{
        unsigned short  opcode;             /* verb operation code          */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   format;             /* format                       */
        unsigned short  primary_rc;         /* primary return code          */
        unsigned long   secondary_rc;       /* secondary return code        */
        unsigned char   *buf_ptr;           /* pointer to buffer            */
        unsigned long   buf_size;           /* buffer size                  */
        unsigned long   total_buf_size;     /* total buffer size required   */
        unsigned short  num_entries;        /* number of entries            */
        unsigned short  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
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   mode_name[8];       /* mode name                    */
        unsigned char   description[RD_LEN];
                                            /* resource description         */
        unsigned short  sess_limit;         /* current session limit        */
        unsigned short  act_sess_count;     /* curr active sessions count   */
        unsigned char   fqplu_name[17];     /* partner LU name              */
        unsigned char   reserv1[3];         /* reserved                     */
} MODE_SUMMARY;
typedef struct mode_detail
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   mode_name[8];       /* mode name                    */
        unsigned char   description[RD_LEN];
                                            /* resource description         */
        unsigned short  sess_limit;         /* session limit                */
        unsigned short  act_sess_count;     /* currently active sess count  */
        unsigned char   fqplu_name[17];     /* partner LU name              */
        unsigned char   reserv1[3];         /* reserved                     */
        unsigned short  min_conwinners_source;
                                            /* min conwinner sess limit     */
        unsigned short  min_conwinners_target;
                                            /* min conloser limit           */
        unsigned char   drain_source;       /* drain source?                */
        unsigned char   drain_partner;      /* drain partner?               */
        unsigned short  auto_act;           /* auto activated conwinner     */
                                            /* session limit                */
        unsigned short  act_cw_count;       /* active conwinner sess count  */
        unsigned short  act_cl_count;       /* active conloser sess count   */
        unsigned char   sync_level;         /* synchronization level        */
        unsigned char   default_ru_size;    /* default RU size to maximize  */
                                            /* performance                  */
        unsigned short  max_neg_sess_limit; /* max negotiated session limit */
        unsigned short  max_rcv_ru_size;    /* max receive RU size          */
        unsigned short  pending_session_count;
                                            /* pending sess count for mode  */
        unsigned short  termination_count;  /* termination count for mode   */
        unsigned char   implicit;           /* implicit or explicit entry   */
        unsigned char   reserva[15];        /* reserved                     */
} MODE_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_MODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros), **plu_alias** (or **fqplu_name** if the **plu_alias** is set to all zeros) and **mode_name** specified (see the following parameters, **lu_name**, **plu_alias**, and **mode_name**) represents an index value that is used to specify the starting point of the actual information to be returned . When a partner LU index is specified, information about other partner LUs is included in the list, if possible.

AP_FIRST_IN_LIST
If **plu_alias** and **fqplu_name** are set to all zeros, the returned list starts from the first partner LU in the list, and the **mode_name** index is ignored. If either **plu_alias** or **fqplu_name** is specified, the list starts at this index, but the **mode_name** index value is ignored, and the returned list starts from the first mode entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** are set to all zeros then the LU associated with the control point (the default LU) is used.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field will be used for determining the index.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

active_sessions
Active session filter. Specifies whether the returned modes should be filtered according to whether they currently have any active sessions (AP_YES or AP_NO).

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

mode_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mode_summary.mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

mode_summary.description
Resource description (as specified on DEFINE_MODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

mode_summary.sess_limit
Current session limit.

mode_summary.act_sess_count
Total number of active sessions using the mode. If the **active_sessions** filter has been set to AP_YES, then this field will always be greater than zero.

mode_summary.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mode_detail.mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

mode_detail.description
Resource description (as specified on DEFINE_MODE).

mode_detail.sess_limit
Current session limit.

mode_detail.act_sess_count
Total number of active sessions using the mode. If the **active_sessions** filter has been set to AP_YES, then this field will always be greater than zero.

mode_detail.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_detail.min_conwinners_source
Specifies the minimum number of sessions on which the local LU is the contention winner (or first speaker).

mode_detail.min_conwinners_target
Specifies the minimum number of sessions on which the local LU is the contention loser (or bidder).

mode_detail.drain_source
Specifies whether the local LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

mode_detail.drain_partner
Specifies whether the partner LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

mode_detail.auto_act
Number of contention winner sessions that are automatically activated following the Change Number of Sessions exchange with the partner LU.

mode_detail.act_cw_count
Number of active, contention winner (or first speaker) 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 (or bidder) sessions using this mode. (The local LU must bid before using one of these sessions.)

mode_detail.sync_level
Specifies the synchronization levels supported by the mode (AP_NONE, AP_CONFIRM, or AP_SYNCPT).

mode_detail.default_ru_size
Specifies whether a default upper bound for the maximum RU size will be used. If this parameter has a value of AP_YES, the **mode_chars.max_ru_size_upp** field specified on **define_mode** is ignored, and the upper bound for the maximum RU size is set to the link BTU size minus the size of the TH and the RH.

AP_YES
AP_NO

mode_detail.max_neg_sess_limit
Maximum negotiable session limit. Specifies the maximum session limit for the mode name that a local LU can use 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 caused the mode session limit to be reset to zero, there might have been conversations using, or waiting to use these sessions. This field is a count of how many of these sessions have not yet been deactivated.

mode_detail.implicit
Specifies whether the entry was put in place because of an implicit (AP_YES) or explicit (AP_NO) definition.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MODE_NAME

AP_INVALID_PLU_NAME
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_MODE_DEFINITION

QUERY_MODE_DEFINITION returns both information previously passed in on a DEFINE_MODE verb and information about SNA-defined default modes.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific mode, or to obtain the list information in several chunks, the **mode_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **mode_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering).

If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

This verb returns definition information only. The QUERY_MODE verb returns information that is determined once the mode starts to be used by a local LU with a partner LU.

VCB Structure

```
typedef struct query_mode_definition
{
        unsigned short  opcode;             /* verb operation code         */
        unsigned char   reserv2;            /* reserved                    */
        unsigned char   format;             /* format                      */
        unsigned short  primary_rc;         /* primary return code         */
        unsigned long   secondary_rc;       /* secondary return code       */
        unsigned char   *buf_ptr;           /* pointer to buffer           */
        unsigned long   buf_size;           /* buffer size                 */
        unsigned long   total_buf_size;     /* total buffer size required  */
        unsigned short  num_entries;        /* number of entries           */
        unsigned short  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
{
        unsigned short  overlay_size;       /* size of this entry          */
        unsigned char   mode_name[8];       /* mode name                   */
        unsigned char   description[RD_LEN];
                                            /* resource description        */
} MODE_DEF_SUMMARY;

typedef struct mode_def_detail
{
        unsigned short  overlay_size;       /* size of this entry          */
        unsigned char   mode_name[8];       /* mode name                   */
        MODE_CHARS      mode_chars;         /* mode characteristics        */
} MODE_DEF_DETAIL;

typedef struct mode_chars
{
        unsigned char   description[RD_LEN];
                                            /* resource description        */
        unsigned short  max_ru_size_upp;    /* max RU size upper bound     */
        unsigned char   receive_pacing_win; /* receive pacing window       */
        unsigned char   default_ru_size;    /* default RU size to maximize */
                                            /* performance                 */
        unsigned short  max_neg_sess_lim;   /* max negotiable session limit */
        unsigned short  plu_mode_session_limit;
                                            /* LU-mode session limit        */
        unsigned short  min_conwin_src;     /* min source contention winner */
                                            /* sessions                     */
        unsigned char   cos_name[8];        /* class-of-service name        */
        unsigned char   cryptography;       /* cryptography                 */
        unsigned char   compression;        /* compression                  */
        unsigned short  auto_act;           /* initial auto-activation count*/
        unsigned short  min_conloser_src;   /* min source contention loser  */
        unsigned short  max_ru_size_low     /* maximum RU size lower bound  */
        unsigned short  max_receive_pacing_win;
                                            /* maximum receive pacing window*/
} MODE_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_MODE_DEFINITION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **mode_name** specified (see the following parameter, **mode_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting

with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

mode_def_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mode_def_summary.mode_name
8-byte mode name, which designates the network properties for a group of sessions.

mode_def_summary.description
Resource description (as specified on DEFINE_MODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

mode_def_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mode_def_detail.mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

mode_def_detail.mode_chars.description
Resource description (as specified on DEFINE_MODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

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.

mode_def_detail.mode_chars.receive_pacing_win
Specifies the session pacing window for the sessions when fixed pacing is used. Specifies the preferred minimum window size when adaptive pacing is used.

mode_def_detail.mode_chars.default_ru_size
Specifies whether a default upper bound for the maximum RU size will be used. If this parameter specifies AP_YES, **max_ru_size_upp** is ignored.

AP_YES
AP_NO

mode_def_detail.mode_chars.max_neg_sess_lim
Maximum negotiable session limit. Value used to negotiate the maximum number of sessions permissible between the local LU and the partner LU for the designated mode name.

mode_def_detail.mode_chars.plu_mode_session_limit
Session limit to negotiate initially on this mode. This value indicates a preferred session limit and is used for implicit CNOS.

Range: 0–32767

mode_def_detail.mode_chars.min_conwin_src
Minimum number of contention winner sessions that can be activated by local LU using this mode.

Range: 0–32767

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

mode_def_detail.mode_chars.cryptography
Specifies whether cryptography is used on sessions using this mode (AP_NONE or AP_MANDATORY).

mode_def_detail.mode_chars.compression
Specifies the use of compression for sessions activated using this mode.

AP_COMP_PROHIBITED
RLE compression is not supported on sessions for this mode.

AP_COMP_REQUESTED
RLE compression is supported and requested (but not mandated) on sessions for this mode.

mode_def_detail.mode_chars.auto_act
Specifies the number of session to be auto-activated for this mode. The value is used for implicit CNOS.

Range: 0–32767

mode_def_detail.mode_chars.min_consloser_src
Specifies the minimum number of contention loser sessions to be activated by any one local LU for this mode. This value is used when CNOS (change number of sessions) exchange is initiated implicitly.

Range: 0–32767

mode_def_detail.mode_chars.max_ru_size_low
Specifies the lower bound for the maximum size of RUs sent and received on sessions in this mode. This value is used when the maximum RU size is negotiated during session activation.

Range: 0–61140

The field is ignored if **default_ru_size** is set to AP_YES.

mode_def_detail.mode_chars.max_receive_pacing_win
Specifies the maximum pacing window for sessions in this mode. For

adaptive pacing, this value is used to limit the receive pacing window it grants. For fixed pacing, this field is not used.

Note: The Program always uses adaptive pacing unless the adjacent node specifies that it does not support it.

Range: 0–32767

The value of zero means that there is no upper bound.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MODE_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_MODE_TO_COS_MAPPING

QUERY_MODE_TO_COS_MAPPING returns information about the mode to COS mapping.

The information is returned as a formatted list. To obtain information about a specific mode, or to obtain the list information in several chunks, the **mode_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered by the **mode_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM’s 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

If the default COS (which unknown modes are mapped to) has been overridden using DEFINE_MODE, QUERY_MODE_TO_COS_MAPPING also returns an entry with null **mode_name** (all zeros) and the default COS. This entry is first in the ordering.

VCB Structure

```
typedef struct query_mode_to_cos_mapping
{
        unsigned short  opcode;            /* verb operation code       */
        unsigned char   reserv2;           /* reserved                  */
        unsigned char   format;            /* format                    */
        unsigned short  primary_rc;        /* primary return code       */
        unsigned long   secondary_rc;      /* secondary return code     */
        unsigned char   *buf_ptr;          /* pointer to buffer         */
        unsigned long   buf_size;          /* buffer size               */
        unsigned long   total_buf_size;    /* total buffer size required */
        unsigned short  num_entries;       /* number of entries         */
        unsigned short  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
{
        unsigned short  overlay_size;      /* size of this 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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **mode_name** specified (see the following parameter, **mode_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This can be set to all zeros to indicate the entry for the default COS.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

mode_to_cos_mapping_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

mode_to_cos_mapping_data.mode_name
8-byte mode name, which designates the network properties for a group of sessions. If this is set to all zeros, it indicates the entry for the default COS.

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

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_MODE_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_NMVT_APPLICATION

QUERY_NMVT_APPLICATION returns a list of applications that have registered for network management vector transport (NMVT) level messages by previously issuing the REGISTER_NMVT_APPLICATION verb (see Chapter 15, “Management Services Verbs”, on page 611 for more details).

The information is returned as a list. To obtain information about a specific application, or to obtain the list information in several chunks, the **application** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

VCB Structure

```
typedef struct query_nmvt_application
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  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
{
        unsigned short  overlay_size;      /* size of this entry           */
        unsigned char   application[8];    /* application name             */
        unsigned short  ms_vector_key_type; /* MS vector key accepted      */
                                           /* by appl                      */
        unsigned char   conversion_required;
                                           /* 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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The **application** specified (see the following parameter, **application**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

application
Application name. The name is an 8-byte alphanumeric type-A EBCDIC character string or all EBCDIC zeros. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

nmvt_application_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nmvt_application_data.application
Name of registered application. The name is an 8-byte alphanumeric type-A EBCDIC character string.

nmvt_application_data.ms_vector_key_type
Management services vector key accepted by the application. When the application registers for NMVT messages, it specifies which management

services vector keys it will accept. For more information on NMVT application registration see Chapter 15, “Management Services Verbs”, on page 611.

nmvt_application_data.conversion_required
Specifies whether the registered application requires messages to be converted from NMVT to MDS_MU format (AP_YES or AP_NO). When the application registers for NMVT messages, it will specify whether this conversion is required. For more information on NMVT application registration, see Chapter 15, “Management Services Verbs”, on page 611.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_APPLICATION_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_NN_TOPOLOGY_NODE

This verb applies only to Communications Server.

Each network node maintains a network topology database that holds information about the network nodes, VRNs and network-node-to-network-node TGs in the network.

QUERY_NN_TOPOLOGY_NODE returns information about the network node and VRN entries in this database.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific node or to obtain the list information in several chunks, the **node_name**, **node_type** and **frsn** fields should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), these fields are ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is by **node_name**, **node_type**, and **frsn**. The **node_name** is ordered by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). The **node_type** field follows the order: AP_NETWORK_NODE, AP_VRN. The **frsn** is ordered numerically.

If AP_LIST_INCLUSIVE is selected, the returned list starts from the first valid record of that name.

If AP_LIST_FROM_NEXT is selected, the list will begin from the first valid record with a name following the one specified.

If the **frsn** field (flow reduction sequence number) is set to a nonzero value, then only database entries with FRSNs higher than this are returned. This allows a consistent topology database to be returned in a number of chunks by first getting the node's current FRSN. This would work as follows:
1. Issue QUERY_NODE, which returns node's current FRSN.
2. Issue as many QUERY_NN_TOPOLOGY_NODE (with FRSN set to zero) as necessary to get all the database entries in chunks.
3. Issue QUERY_NODE again and compare the new FRSN with the one returned in step 1.
4. If the two FRSNs are different, then the database has changed, so issue a QUERY_NN_TOPOLOGY_NODE with the FRSN set to 1 greater than the FRSN supplied in step 1.

VCB Structure

```
typedef struct query_nn_topology_node
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  total_num_entries; /* total number of entries      */
        unsigned char   list_options;      /* listing options              */
        unsigned char   reserv3;           /* reserved                     */
        unsigned char   node_name[17];     /* network qualified node name  */
        unsigned char   node_type;         /* node type                    */
        unsigned long   frsn;              /* flow reduction sequence num  */
} QUERY_NN_TOPOLOGY_NODE;
```

Note: If the **frsn** field is set to a nonzero value, then only node entries with FRSNs greater than the one specified are returned. If it is set to zero, then all node entries are returned.

```
typedef struct nn_topology_node_summary
{
        unsigned short  overlay_size;      /* size of this entry           */
        unsigned char   node_name[17];     /* network qualified node name  */
        unsigned char   node_type;         /* node type                    */
} NN_TOPOLOGY_NODE_SUMMARY;
typedef struct nn_topology_node_detail
{
        unsigned short  overlay_size;      /* size of this entry           */
        unsigned char   node_name[17];     /* network qualified node name  */
        unsigned char   node_type;         /* node type                    */
        unsigned short  days_left;         /* days left until entry purged */
        unsigned char   reserv1[2];        /* reserved                     */
        unsigned long   frsn;              /* flow reduction sequence num  */
        unsigned long   rsn;               /* resource sequence number     */
        unsigned char   rar;               /* route additional resistance  */
        unsigned char   status;            /* node status                  */
        unsigned char   function_support;  /* function support             */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   branch_aware;      /* node is branch aware         */
        unsigned char   reserva[20];       /* reserved                     */
} NN_TOPOLOGY_NODE_DETAIL;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_NN_TOPOLOGY_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **node_name**, **node_type**, and **frsn** specified (see the following parameters, **node_name**, **node_type**, and **frsn**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

node_name
Network qualified node name from network topology database. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

node_type
Type of the node. This can be one of the following values:

AP_NETWORK_NODE
AP_VRN

If the **node_type** is unknown, AP_LEARN_NODE must be specified.

frsn Flow Reduction Sequence Number. If this is nonzero, then only nodes with a FRSN greater than or equal to this value are returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

nn_topology_node_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nn_topology_node_summary.node_name
Network qualified node name from network topology database. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

nn_topology_node_summary.node_type
Type of the node. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

nn_topology_node_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nn_topology_node_detail.node_name
Network qualified node name from network topology database. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

nn_topology_node_detail.node_type
Type of the node. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

nn_topology_node_detail.days_left
Number of days before deletion of this node entry from the topology database. This will be set to zero for the local node entry (this entry is never deleted).

nn_topology_node_detail.frsn
Flow Reduction Sequence Number. It indicates the last time that this resource was updated at the local node.

nn_topology_node_detail.rsn
Resource Sequence Number. This is assigned by the network node that owns this resource.

nn_topology_node_detail.rar
The node's route additional resistance.

nn_topology_node_detail.status
Specifies the status of the node. This can be AP_UNCONGESTED or one or more of the following values ORed together:

AP_CONGESTED
The number of ISR sessions is greater than the **isr_sessions_upper_threshold**.

AP_ERR_DEPLETED
The number of endpoint sessions has reached the maximum specified.

AP_IRR_DEPLETED
The number of ISR sessions has reached the maximum.

AP_QUIESCING
A STOP_NODE or type AP_QUIESCE or AP_QUIESCE_ISR has been issued

nn_topology_node_detail.function_support
Specifies which functions are supported. This can be one or more of the following values:

AP_PERIPHERAL BORDER_NODE
Peripheral Border Node function is supported.

AP_EXTENDED BORDER_NODE
Extended Border Node function is supported.

AP_CDS
Node supports central directory server function.

AP_GATEWAY
Node is a gateway Node. (This function is not yet architecturally defined.)

AP_INTERCHANGE_NODE
This node is a Gateway Node. (This function is not yet architecturally defined.)

AP_ISR
Node supports intermediate session routing.

AP_HPR
Node supports the base functions of High-Performance Routing.

AP_RTP_TOWER
Node supports the RTP tower of HPR.

AP_CONTROL_OVER_RTP_TOWER
Node supports the control flows over the RTP tower.

Note: The AP_CONTROL_OVER_RTP_TOWER corresponds to the setting of both AP_HPR and AP_RTP_TOWER.

nn_topology_node_detail.branch_aware
Specifies whether the node is branch aware.

AP_NO
The node is not branch aware.

AP_YES
The node is branch aware.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_NODE

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_NN_TOPOLOGY_STATS

This verb applies only to Communications Server.

QUERY_NN_TOPOLOGY_STATS returns statistical information about the topology database and is only issued at a network node.

VCB Structure

```
typedef struct query_nn_topology_stats
{
        unsigned short  opcode;             /* verb operation code           */
        unsigned char   reserv2;            /* reserved                      */
        unsigned char   format;             /* format                        */
        unsigned short  primary_rc;         /* primary return code           */
        unsigned long   secondary_rc;       /* secondary return code         */
        unsigned long   max_nodes;          /* max num of nodes in database  */
        unsigned long   cur_num_nodes;      /* current number of nodes in    */
                                            /* database                      */
        unsigned long   node_in_tdus;       /* number of TDUs received       */
        unsigned long   node_out_tdus;      /* number of TDUs sent           */
        unsigned long   node_low_rsns;      /* node updates received with    */
                                            /* low RSNs                      */
        unsigned long   node_equal_rsns;    /* node updates in with          */
                                            /* equal RSNs                    */
        unsigned long   node_good_high_rsns;
                                            /* node updates in with          */
                                            /* high RSNs                     */
        unsigned long   node_bad_high_rsns;
                                            /* node updates in with          */
                                            /* high and odd RSNs             */
        unsigned long   node_state_updates; /* number of node updates sent   */
        unsigned long   node_errors;        /* number of node entry          */
                                            /* errors found                  */
        unsigned long   node_timer_updates; /* number of node records built  */
                                            /* due to timer updates          */
        unsigned long   node_purges;        /* num node records purged       */
        unsigned long   tg_low_rsns;        /* TG updates received with      */
                                            /* low RSNs                      */
        unsigned long   tg_equal_rsns;      /* TG updates in with equal RSNs */
        unsigned long   tg_good_high_rsns;  /* TG updates in with high RSNs  */
        unsigned long   tg_bad_high_rsns;   /* TG updates in with high       */
                                            /* and odd RSNs                  */
        unsigned long   tg_state_updates;   /* number of TG updates sent     */
        unsigned long   tg_errors;          /* number of TG entry errors     */
                                            /* found                         */
        unsigned long   tg_timer_updates;   /* number of node records        */
                                            /* built due to timer updates    */
        unsigned long   tg_purges;          /* num node records purged       */
        unsigned long   total_route_calcs;  /* num routes calculated for COS */
        unsigned long   total_route_rejs;   /* num failed route calculations */
        unsigned long   total_tree_cache_hits;
                                            /* total num of tree cache hits  */
        unsigned long   total_tree_cache_misses;
                                            /* total num of tree cache       */
                                            /* misses                        */
        unsigned counter
                        total_tdu_wars;    /* total number TDU war           */
        unsigned char   reserva[16];       /* reserved                     */
} QUERY_NN_TOPOLOGY_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_NN_TOPOLOGY_STATS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

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

primary_rc
AP_OK

max_nodes
Maximum number of node records in the topology database (zero means unlimited).

cur_num_nodes
Current number of nodes in this node's topology database. If this value exceeds the maximum number of nodes allowed, an Alert is issued.

node_in_tdus
Total number of topology database updates (TDUs) received by this node.

node_out_tdus
Total number of topology database updates (TDUs) built by this node to be sent to all adjacent network nodes since the last initialization.

node_low_rsns
Total number of topology node updates received by this node with an RSN less than the current RSN. Both even and odd RSNs are included in this count. (These TDUs are not errors, but result when TDUs are broadcast to all adjacent network nodes. No update to this node's topology database occurs, but this node sends a TDU with its higher RSN to the adjacent node that sent this low RSN.)

node_equal_rsns
Total number of topology node updates received by this node with an RSN equal to the current RSN. Both even and odd RSNS are included in this count. (These TDUs are not errors, but result when TDUs are broadcast to all adjacent network nodes. No update to this node's topology database occurs.)

node_good_high_rsns
Total number of topology node updates received by this node with an RSN greater than the current RSN. The node updates its topology and broadcasts a TDU to all adjacent network nodes. It is not required to send a TDU to the sender of this update, because that node already has the update.

node_bad_high_rsns
Total number of topology node updates received by this node with an odd RSN greater than the current RSN. These updates represent a topology inconsistency detected by one of the APPN network nodes. The node updates its topology and broadcasts a TDU to all adjacent network nodes.

node_state_updates
Total number of topology node updates built as a result of internally

detected node state changes that affect APPN topology and routing. Updates are sent by TDUs to all adjacent network nodes.

node_errors
Total number of topology node update inconsistencies detected by this node. This occurs when this node attempts to update its topology database and detects a data inconsistency. This node creates a TDU with the current RSN incremented to the next odd number and broadcasts it to all adjacent network nodes.

node_timer_updates
Total number of topology node updates built for this node's resource due to timer updates. Updates are sent by TDUs to all adjacent network nodes. These updates ensure that other network nodes do not delete this node's resource from their topology database.

node_purges
Total number of topology node records purged from this node's topology database. This occurs when a node record has not been updated in a specified amount of time. The owning node is responsible for broadcasting updates for its resource that it wants kept in the network topology.

tg_low_rsns
Total number of topology TG updates received by this node with an RSN less than the current RSN. Both even and odd RSNs are included in this count. (These TDUs are not errors, but result when TDUs are broadcast to all adjacent network nodes. No update to this node's topology database occurs, but this node sends a TDU with its higher RSN to the adjacent node that sent this low RSN.)

tg_equal_rsns
Total number of topology TG updates received by this node with an RSN equal to the current RSN. Both even and odd RSNs are included in this count. (These TDUs are not errors, but result when TDUs are broadcast to all adjacent network nodes. No update to this node's topology database occurs.)

tg_good_high_rsns
Total number of topology TG updates received by this node with an RSN greater than the current RSN. The node updates its topology and broadcasts a TDU to all adjacent network nodes.

tg_bad_high_rsns
Total number of topology TG updates received by this node with an odd RSN greater than the current RSN. These updates represent a topology inconsistency detected by one of the APPN Network Nodes. The node updates its topology and broadcasts a TDU to all adjacent network nodes.

tg_state_updates
Total number of topology TG updates built as a result of internally detected node state changes that affect APPN topology and routing. Updates are sent by TDUs to all adjacent network nodes.

tg_errors
Total number of topology TG update inconsistencies detected by this node. This occurs when this node attempts to update its topology database and detects a data inconsistency. This node creates a TDU with the current RSN incremented to the next odd number and broadcasts it to all adjacent network nodes.

tg_timer_updates
Total number of topology TG updates built for this node's resource due to timer updates. Updates are sent by TDUs to all adjacent network nodes. These updates ensure that other network nodes do not delete this node's resource from their topology database.

tg_purges
Total number of topology TG records purged from this node's topology database. This occurs when a node record has not been updated in a specified amount of time. The owning node is responsible for broadcasting updates for its resource that it wants kept in the network topology.

total_route_calcs
Number of routes calculated for all classes of service since the last.

total_route_rejs
Number of route requests for all classes of service that could not be calculated since the last initialization.

total_tree_cache_hits
Number of route computations that were satisfied by a cached routing tree. Note that this number may be greater than the total number of computed routes, because each route may require inspection of several trees.

total_tree_cache_misses
Number of route computations that were not satisfied by a cached routing tree, so that a new routing tree had to be built.

total_tdu_wars
Number of TDU wars the local node has detected and prevented.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_NN_TOPOLOGY_TG

This verb applies only to Communications Server.

Each network node maintains a network topology database which holds information about the network nodes, VRNs and network-node-to-network-node TGs in the network. QUERY_NN_TOPOLOGY_TG returns information about the TG entries in this database.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific node or to obtain the list information in several chunks, the **owner**, **owner_type**, **dest**, **dest_type**, **tg_num**, and **frsn** fields should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), these fields are ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is by **owner**, **owner_type**, **dest**, **dest_type**, **tg_num**, and **frsn**. The **owner** name and **dest** name are ordered by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). The **owner_type** and **dest_type** follow the order: AP_NETWORK_NODE, AP_VRN. The **tg_num** and **frsn** are ordered numerically.

If AP_LIST_INCLUSIVE is selected, the returned list starts from the first valid record of that name.

If AP_LIST_FROM_NEXT is selected, the list will begin from the first valid record with a name following the one specified.

If the **frsn** field (flow reduction sequence number) is set to a nonzero value, then only database entries with FRSNs higher than this are returned. This allows a consistent topology database to be returned in a number of chunks by first getting the node's current FRSN. This works as follows:

1. Issue QUERY_NODE, which returns the node's current FRSN.
2. Issue as many QUERY_NN_TOPOLOGY_TG (with FRSN set to zero) as necessary to get all the database entries in chunks.
3. Issue QUERY_NODE again and compare the new FRSN with the one returned in step 1.
4. If the two FRSNs are different, then the database has changed, so issue a QUERY_NN_TOPOLOGY_TG with the FRSN set to 1 greater than the FRSN supplied in step 1.

VCB Structure

```
typedef struct query_nn_topology_tg
{
        unsigned short  opcode;           /* verb operation code          */
        unsigned char   reserv2;          /* reserved                     */
        unsigned char   format;           /* format                       */
        unsigned short  primary_rc;       /* primary return code          */
        unsigned long   secondary_rc;     /* secondary return code        */
        unsigned char   *buf_ptr;         /* pointer to buffer            */
        unsigned long   buf_size;         /* buffer size                  */
        unsigned long   total_buf_size;   /* total buffer size required   */
        unsigned short  num_entries;      /* number of entries            */
        unsigned short  total_num_entries; /* total number of entries     */
        unsigned char   list_options;     /* listing options              */
        unsigned char   reserv3;          /* reserved                   */
        unsigned char   owner[17];        /* node that owns the TG      */
        unsigned char   owner_type;       /* type of node that owns the TG*/
        unsigned char   dest[17];         /* TG destination node        */
        unsigned char   dest_type;        /* TG destination node type   */
        unsigned char   tg_num;           /* TG number                  */
        unsigned char   reserv1;          /* reserved                   */
        unsigned long   frsn;             /* flow reduction sequence num  */
} QUERY_NN_TOPOLOGY_TG;

typedef struct topology_tg_summary
{
        unsigned short  overlay_size;     /* size of this entry         */
        unsigned char   owner[17];        /* node that owns the TG      */
        unsigned char   owner_type;       /* type of node that owns the TG*/
        unsigned char   dest[17];         /* TG destination node        */
        unsigned char   dest_type;        /* TG destination node type   */
        unsigned char   tg_num;           /* TG number                  */
        unsigned char   reserv3[1];       /* reserved                   */
        unsigned long   frsn;             /* flow reduction sequence num  */
} TOPOLOGY_TG_SUMMARY;

typedef struct topology_tg_detail
{
        unsigned short  overlay_size;     /* size of this entry         */
        unsigned char   owner[17];        /* node that owns the TG      */
        unsigned char   owner_type;       /* type of node that owns the TG*/
        unsigned char   dest[17];         /* TG destination node        */
        unsigned char   dest_type;        /* TG destination node type   */
        unsigned char   tg_num;           /* TG number                  */
        unsigned char   reserv3[1];       /* reserved                   */
        unsigned long   frsn;             /* flow reduction sequence num  */
        unsigned short  days_left;        /* days left until entry purged */
        LINK_ADDRESS    dlc_data          /* DLC signalling data        */
        unsigned long   rsn;              /* resource sequence number   */
        unsigned char   status;           /* node status                */
        TG_DEFINED_CHARS tg_chars;        /* TG characteristics         */
        unsigned char   subarea_number[4];
                                          /* subarea number             */
        unsigned char   tg_type;          /* TG type                    */
        unsigned char   intersubnet_tg;   /* intersubnet TG             */
        unsigned char   cp_cp_session_active;
                                          /* CP-CP session is active    */
        unsigned char   branch_tg;        /* TG is a branch TG          */
        unsigned char   reserva[12];      /* reserved                   */
} TOPOLOGY_TG_DETAIL;

typedef struct link_address
{
        unsigned short  length;            /* length                    */
        unsigned short reserve1;           /* reserved                  */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                           /* address                   */
} LINK_ADDRESS;
```

Note: If the **frsn** field is set to a nonzero value, then only node entries with that FRSN are returned. If it is set to zero, then all node entries are returned.

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_NN_TOPOLOGY_TG

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **owner**, **owner_type**, **dest**, **dest_type**, **tg_num**, and **frsn** specified (see the following parameters, **owner**, **owner_type**, **dest**, **dest_type**, **tg_num**, and **frsn**) represents an index value that is used to specify the starting point of the actual information to be returned .

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

owner Name of the TG's originating node. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

owner_type
Type of the node that owns the TG. This can be one of the following values:

AP_NETWORK_NODE
AP_VRN

If the **owner_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dest Fully qualified destination node name for the TG. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dest_type
Type of the destination node for this TG. This can be one of the following values:

AP_NETWORK_NODE
AP_VRN

If the **dest_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

tg_num
Number associated with the TG. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

frsn Flow Reduction Sequence Number. If this is nonzero, then only nodes with a FRSN greater than or equal to this value are returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

topology_tg_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

topology_tg_summary.owner
Name of the TG's originating node. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

topology_tg_summary.owner_type
Type of the node that owns the TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

topology_tg_summary.dest
Fully qualified destination node name for the TG. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

topology_tg_summary.dest_type
Type of the destination node for this TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

topology_tg_summary.tg_num
Number associated with the TG.

topology_tg_summary.frsn
Flow Reduction Sequence Number. It indicates the last time that this resource was updated at the local node.

topology_tg_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

topology_tg_detail.owner
Name of the TG's originating node. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

topology_tg_detail.owner_type
Type of the node that owns the TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

topology_tg_detail.dest
Fully qualified destination node name for the TG. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

topology_tg_detail.dest_type
Type of the destination node for this TG. This is set to one of the following values:

AP_NETWORK_NODE
AP_VRN

topology_tg_detail.tg_num
Number associated with the TG.

topology_tg_detail.frsn
Flow Reduction Sequence Number. It indicates the last time that this resource was updated at the local node.

topology_node_detail.days_left
Number of days before deletion of this node entry from the topology database.

topology_tg_detail.dlc_data.length
Length of DLC address of connection to a VRN (set to zero if **dest_type** is not AP_VRN).

topology_tg_detail.dlc_data.address
DLC address of connection to VRN. This is set to zero if **dest_type** is not AP_VRN.

topology_tg_detail.rsn
Resource Sequence Number. This is assigned by the network node that owns this resource.

topology_tg_detail.status
Specifies the status of the TG. This can be one or more of the following values ORed together:

AP_TG_OPERATIVE
AP_TG_QUIESCING
AP_TG_GARBAGE_COLLECT
AP_TG_CP_CP_SESSIONS
AP_TG_HPR
AP_TG_RTP
AP_TG_NONE

topology_tg_detail.tg_chars
TG characteristics (See "DEFINE_COS" on page 35).

topology_tg_detail.subarea_number
If the owner or destination node of the TG is subarea-capable, this field contains the subarea number of the type 4 or type 5 node that owns the link station associated with this TG on the subarea-capable node. Otherwise, this field is set to all binary zeros.

topology_tg_detail.tg_type
TG type. This field takes one of the following values:

AP_APPN_OR_BOUNDARY_TG
APPN TG or boundary-function-based TG

AP_INTERCHANGE_TG
Interchange TG

AP_VIRTUAL_ROUTE_BASED_TG
Virtual-route-based TG

AP_UNKNOWN
The TG type of this TG reported in the topology is unknown.

topology_tg_detail.intersubnet.tg
This TG is an intersubnetwork TG. This field takes the following values:

AP_YES
AP_NO

topology_tg_detail.cp_cp_session_active
Specifies whether the owning node's contention winner CP-CP session is active (AP_UNKNOWN, AP_NO or AP_YES).

topology_tg_detail.branch_tg
Specifies whether the TG is a branch TG.

AP_NO
The TG is not a branch TG.

AP_YES
The TG is a branch TG.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TG

AP_INVALID_ORIGIN_NODE
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_NODE

QUERY_NODE returns node specific information and statistics. In addition to returning information determined dynamically during execution, QUERY_NODE also returns parameters which are set during node initialization.

VCB Structure

Format 2

```
typedef struct query_node
{
        unsigned short  opcode;            /* verb operation code         */
        unsigned char   reserv2;           /* reserved                    */
        unsigned char   format;            /* format                      */
        unsigned short  primary_rc;        /* primary return code         */
        unsigned long   secondary_rc;      /* secondary return code       */
        CP_CREATE_PARMS cp_create_parms;   /* create parameters           */
        unsigned long   up_time;           /* time since node started     */
        unsigned long   mem_size;          /* size of memory available    */
        unsigned long   mem_used;          /* size of memory used         */
        unsigned long   mem_warning_threshold;
                                           /* memory constrained          */
                                           /* threshold                   */
        unsigned long   mem_critical_threshold;
                                           /* memory critical threshold   */
        unsigned char   nn_functions_supported;
                                           /* NN functions supported      */
        unsigned char   functions_supported;
                                           /* functions supported         */
        unsigned char   en_functions_supported;
                                           /* EN functions supported      */
        unsigned char   nn_status;         /* node status.  One or more of */
        unsigned long   nn_frsn;           /* NN flow reduction           */
                                           /* sequence number             */
        unsigned long   nn_rsn;            /* Resource sequence number    */
        unsigned short  def_ls_good_xids;  /* Good XIDs for defined       */
                                           /* link stations               */
        unsigned short  def_ls_bad_xids;   /* Bad XIDs for defined        */
                                           /* link stations               */
        unsigned short  dyn_ls_good_xids;  /* Good XIDs for dynamic       */
                                           /* link stations               */
        unsigned short  dyn_ls_bad_xids;   /* Bad XIDs for dynamic        */
                                           /* link stations               */
        unsigned char   dlur_release_level; /* Current DLUR release level */
        unsigned char   nns_dlus_served_lu_reg_supp;
                                           /* NNS support for registration */
                                           /* of DLUS-served LUs reserved  */
       unsigned char   reserva[19];        /* reserved                    */
       unsigned char   fq_nn_server_name[17];                             */
                                           /* FQ name of NN server        */
       unsigned long   current_isr_sessions;/* current ISR sessions       */
       unsigned char   nn_functions2;      /* NN functions continued      */
       unsigned char   branch_ntwk_arch_version;
                                           /* branch network architecture  */
                                           /* version supported           */
       unsigned char   reservb[28];        /* reserved                    */
} QUERY_NODE;
```

```
typedef struct cp_create_parms
{
        unsigned short  crt_parms_len;     /* length of CP_CREATE_PARMS   */
        unsigned char   description[RD_LEN];
                                           /* resource description        */
        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                      */
        unsigned short  max_locates;        /* max locates node can process */
        unsigned short  dir_cache_size;     /* directory cache size         */
                                            /* (reserved) if not NN)        */
        unsigned short  max_dir_entries;    /* max directory entries        */
        unsigned short  locate_timeout;     /* locate timeout in seconds    */
        unsigned char   reg_with_nn;        /* register resources with NNS  */
        unsigned char   reg_with_cds;       /* resource registration with   */
                                            /* CDS                          */
        unsigned short  mds_send_alert_q_size;
                                            /* size of MDS send alert queue */
        unsigned short  cos_cache_size;     /* number of COS definitions    */
        unsigned short  tree_cache_size;    /* Topology Database routing    */
                                            /* tree cache size              */
        unsigned short  tree_cache_use_limit;
                                            /* num times tree can be used   */
        unsigned short  max_tdm_nodes;      /* max num nodes that can be    */
                                            /* stored in Topology Database  */
        unsigned short  max_tdm_tgs;        /* max num TGs that can be      */
                                            /* stored in Topology Database  */
        unsigned long   max_isr_sessions;   /* max ISR sessions             */
        unsigned long   isr_sessions_upper_threshold;
                                            /* upper threshold for ISR sess */
        unsigned long   isr_sessions_lower_threshold;
                                            /* lower threshold for ISR sess */
        unsigned short  isr_max_ru_size;    /* max RU size for ISR          */
        unsigned short  isr_rcv_pac_window; /* ISR rcv pacing window size   */
        unsigned char   store_endpt_rscvs;  /* endpoint RSCV storage        */
        unsigned char   store_isr_rscvs;    /* ISR RSCV storage             */
        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;        /* level of HPR support         */
        unsigned char   mobile;             /* HPR path-switch controller?  */
        unsigned char   discovery_support;  /* Discovery function utilized  */
        unsigned char   discovery_group_name[8];
                                            /* Group name for Discovery     */
        unsigned char   implicit_lu_0_to_3;
                                            /* Implicit LU 0 to 3 support   */
        unsigned char   default_preference;
                                            /* Default routing preference   */
        unsigned char   anynet_supported;
                                            /* level of AnyNet support      */
        unsigned short  max_ls_exception_events;
                                            /* maximum LS Exception events  */
        unsigned char   comp_in_series;     /* compression in series allowed*/
        unsigned char   max_compress_lvl;   /* maximum compression level    */
        unsigned char   node_spec_data_len; /* length of node specific data */
        unsigned char   ptf[64];            /* program temporary fix array  */
} CP_CREATE_PARMS;
```

Format 1 (back-level)

```
typedef struct query_node
{
        unsigned short  opcode;             /* verb operation code          */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   format;             /* format                       */
        unsigned short  primary_rc;         /* primary return code          */
        unsigned long   secondary_rc;       /* secondary return code        */
        CP_CREATE_PARMS cp_create_parms;    /* create parameters            */
        unsigned long   up_time;            /* time since node started      */
        unsigned long   mem_size;           /* size of memory available     */
        unsigned long   mem_used;           /* size of memory used          */
        unsigned long   mem_warning_threshold;
                                            /* memory constrained           */
                                            /* threshold                    */
        unsigned long   mem_critical_threshold;
                                            /* memory critical threshold    */
        unsigned char   nn_functions_supported;
                                            /* NN functions supported       */
        unsigned char   functions_supported;
                                            /* functions supported          */
        unsigned char   en_functions_supported;
                                            /* EN functions supported       */
        unsigned char   nn_status;          /* node status.  One or more of */
        unsigned long   nn_frsn;            /* NN flow reduction            */
                                            /* sequence number              */
        unsigned long   nn_rsn;             /* Resource sequence number     */
        unsigned short  def_ls_good_xids;   /* Good XIDs for defined        */
                                            /* link stations                */
        unsigned short  def_ls_bad_xids;    /* Bad XIDs for defined         */
                                            /* link stations                */
        unsigned short  dyn_ls_good_xids;   /* Good XIDs for dynamic        */
                                            /* link stations                */
        unsigned short  dyn_ls_bad_xids;    /* Bad XIDs for dynamic         */
                                            /* link stations                */
        unsigned char   dlur_release_level; /* Current DLUR release level   */
        unsigned char   reserva[19];         /* reserved                    */
} QUERY_NODE;
```

Format 0 (back-level)

```
typedef struct query_node
{
        unsigned short  opcode;             /* verb operation code          */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   format;             /* format                       */
        unsigned short  primary_rc;         /* primary return code          */
        unsigned long   secondary_rc;       /* secondary return code        */
        CP_CREATE_PARMS cp_create_parms;    /* create parameters            */
        unsigned long   up_time;            /* time since node started      */
        unsigned long   mem_size;           /* size of memory available     */
        unsigned long   mem_used;           /* size of memory used          */
        unsigned long   mem_warning_threshold;
                                            /* memory constrained           */
                                            /* threshold                    */
        unsigned long   mem_critical_threshold;
                                            /* memory critical threshold    */
        unsigned char   nn_functions_supported;
                                            /* NN functions supported       */
        unsigned char   functions_supported;
                                            /* functions supported          */
        unsigned char   en_functions_supported;
                                            /* EN functions supported       */
        unsigned char   nn_status;          /* node status.  One or more of */
        unsigned long   nn_frsn;            /* NN flow reduction            */
                                            /* sequence number              */
        unsigned long   nn_rsn;             /* Resource sequence number     */
        unsigned short  def_ls_good_xids;   /* Good XIDs for defined        */
                                            /* link stations                */
        unsigned short  def_ls_bad_xids;    /* Bad XIDs for defined         */
                                            /* link stations                */
        unsigned short  dyn_ls_good_xids;   /* Good XIDs for dynamic        */
                                            /* link stations                */
        unsigned short  dyn_ls_bad_xids;    /* Bad XIDs for dynamic         */
```

```
                                             /* link stations              */
        unsigned char   dlur_release_level; /* Current DLUR release level   */
        unsigned char   reserva[19];        /* reserved                     */
} QUERY_NODE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

When this field is set to zero, the following four fields are Unsigned short rather than Unsigned_COUNTER:**def_Is_good_xids**, **def_Is_bad_xids**,**dyn_Is_good_xids**, **dyn_Is_bad_xids**.

When this field is set to two, the following fields are used as described: **fq_nn_server_name** and **current_isr_sessions**.

Returned Parameters

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

primary_rc
AP_OK

cp_create_parms.crt_parms_len
Length of create parameters structure.

cp_create_parms.description
Resource description. This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

cp_create_parms.node_type
This is always:

AP_END_NODE
AP_NETWORK_NODE
AP_LEN_NODE
AP_BRANCH_NETWORK_NODE

cp_create_parms.fqcp_name
Node's 17-byte fully qualified control point name. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name has a maximum length of 8 bytes with no embedded spaces.)

cp_create_parms.cp_alias
Locally used control point alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

cp_create_parms.mode_to_cos_map_supp
Specifies whether mode to COS mapping is supported by the node (AP_YES or AP_NO). If this is set to AP_YES then the COS specified on a DEFINE_MODE verb must either be an SNA defined COS or have been defined by issuing a DEFINE_COS verb.

cp_create_parms.mds_supported
Specifies whether management services supports Multiple Domain Support and Management Services Capabilities (AP_YES or AP_NO).

cp_create_parms.node_id
Node identifier used in XID exchange. This 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
Network node only: Size of the directory cache.

cp_create_parms.max_dir_entries
Maximum number of directory entries. This is unlimited if this field is set to zero.

cp_create_parms.locate_timeout
Specifies the time in seconds before a network search will time out. A value of zero indicates that the search has no timeout.

cp_create_parms.reg_with_nn
Specifies whether resources will be registered with the network node server. Registration failure does not affect successful completion of node initialization. See “REGISTRATION_FAILURE” on page 547 for details. This field is interpreted differently by an EN and a BrNN.

End Node:

AP_NO
The node does not register any LUs with its NN server. The NNS forwards all broadcast searches to the end node.

AP_YES
The node registers all local dependent (if the NNS supports option set 1116) and all local independent LUs with its NNS. The NNS only forwards directed locates to it (unless it owns dependent LUs that could not be registered).

Branch Network Node:

AP_REGISTER_NONE
The node does not register any LUs with its NN server.

AP_REGISTER_ALL
The node registers all local dependent (if it supports DLUR full multi-subnet and the NNS supports option set 1116) and all domain independent LUs with its NNS.

AP_REGISTER_LOCAL_ONLY
The node registers all local dependent (if it supports DLUR full multi-subnet and the NNS supports option set 1116) and all local independent LUs with its NNS.

cp_create_parms.reg_with_cds
Specifies whether resources are allowed to be registered with a central directory server (CDS). This field is interpreted differently by an EN, NN, or BrNN.

End Node: Specifies whether the NNS is allowed to register with CDS end node resources. This field is ignored if **reg_with_nn** is set to AP_NONE.

AP_NO
EN resources cannot be registered with a CDS.

AP_YES
EN resources can be registered with a CDS.

Network Node: Specifies whether local resources and domain resources (that the owning EN allows to be registered with a CDS) can be registered with a CDS.

AP_NO
Local or domain resources cannot be registered with a CDS.

AP_YES
Local or domain resources can be registered with a CDS. Registration failure does not affect successful completion of the START_NODE verb.

Branch Network Node: Specifies whether the NNS is allowed to register with a CDS BrNN resources (local to the BrNN or from the BrNN's domain). This field is ignored if **reg_with_nn** is set to AP_NO.

AP_REGISTER_NONE
The node does not register any LUs with its NN server.

AP_REGISTER_ALL
The node registers all local dependent (if it supports DLUR full multi-subnet and the NNS supports option set 1116) and all domain independent LUs with its NNS.

AP_REGISTER_LOCAL_ONLY
The node registers all local dependent (if it supports DLUR full multi-subnet and the NNS supports option set 1116) and all local independent LUs with its NNS.

cp_create_parms.mds_send_alert_q_size
Size of the MDS send alert queue. When this limit is reached, the MDS component deletes the oldest entry on the queue.

cp_create_parms.cos_cache_size
Size of the COS Database weights cache.

cp_create_parms.tree_cache_size
Size of the topology database routing tree cache size.

cp_create_parms.tree_cache_use_limit
Maximum number of uses of a cached tree. Once this number is exceeded, the tree is discarded and recomputed. This allows the node to balance sessions among equal weight routes. A low value provides better load balancing at the expense of increased activation latency.

cp_create_parms.max_tdm_nodes
Maximum number of nodes that can be stored in topology database (zero means unlimited).

cp_create_parms.max_tdm_tgs
Maximum number of TGs that can be stored in topology database (zero means unlimited).

cp_create_parms.max_isr_sessions
Maximum number of ISR sessions the node can participate in at once.

cp_create_parms.isr_sessions_upper_threshold
See **cp_create_parms.isr_sessions_lower_threshold**

cp_create_parms.isr_sessions_lower_threshold
The upper and lower thresholds control the node's congestion status. The node state changes from not congested to congested if the number of ISR

sessions exceeds the upper threshold. The node state changes back to not congested once the number of ISR sessions dips below the lower threshold.

cp_create_parms.isr_max_ru_size
Maximum RU size supported for intermediate sessions.

cp_create_parms.isr_rcv_pac_window
Suggested receive pacing window size for intermediate sessions. This value is only used on the secondary hop of intermediate sessions if the adjacent node does not support adaptive pacing.

cp_create_parms.store_endpt_rscvs
Specifies whether RSCVs are stored for diagnostic purposes (AP_YES or AP_NO).

cp_create_parms.store_isr_rscvs
Specifies whether RSCVs are stored for diagnostic purposes (AP_YES or AP_NO).

cp_create_parms.store_dlur_rscvs
Specifies whether the node stores RSCVs for diagnostic purposes (AP_YES or AP_NO). If this field is set to AP_YES, then an RSCV is returned on the QUERY_DLUR_LU verb.

cp_create_parms.dlur_support
Specifies the level of support for DLUR provided by the node. This is a bit field and may take the following values:

AP_NO
DLUR is not supported.

AP_YES
DLUR full multi-subnet is supported.

(AP_YES | AP_LIMITED_DLUR_MULTI_SUBNET)
DLUR limited, DLUR multi-subnet is supported. This is only valid if the node is an end node.

cp_create_parms.pu_conc_support
Specifies whether PU concentration is supported (always AP_NO).

cp_create_parms.nn_rar
The network node's route additional resistance.

cp_create_parms.hpr_support
Specifies the level of support for HPR that is provided by the node (AP_NONE, AP_BASE, or AP_RTP).

cp_create_parms.mobile
Specifies whether the node is an HPR path-switch controller (AP_YES or AP_NO). If the **cp_create_parms.hpr_support** field is not set to AP_RTP this field is reserved.

cp_create_parms.discovery_support
Specifies whether Discovery functions are utilized by this node.

AP_DISCOVERY_CLIENT
Discovery client functions are used by this node

AP_DISCOVERY_SERVER
Discovery server functions are used by this node.

cp_create_parms.discovery_group_name
Specifies the group name used on Discovery functions utilized by the node. If this field is set to all zeros, the default group name is used.

cp_create_parms.implicit_lu_0_to_3
Specifies whether the node supports implicit definition of LUs of type 0 to 3 by ACTLU (AP_YES or AP_NO).

cp_create_parms.default_preference
Specifies the preferred method of routing when initiating sessions from this node.

Note: This can be overridden on a per LU basis using the DEFINE_PARTNER_LU verb.
This field can take the following values:

AP_NATIVE
Use native (APPN) routing protocols only.

AP_NONNATIVE
Use nonnative (AnyNet) routing protocols only.

AP_NATIVE_THEN_NONNATIVE
Try native (APPN) protocols, and if the partner LU cannot be located, then retry session activation using nonnative (AnyNet) protocols.

AP_NONNATIVE_THEN_NATIVE
Try nonnative (AnyNet) protocols, and if the partner LU cannot be located, then retry session activation using native (APPN) protocols.

Note: The latter three values are only meaningful when an AnyNet DLC is available to the Node Operator Facility, and there is an AnyNet Link Station defined.

cp_create_parms.anynet_supported
Specifies support for the AnyNet DLC. This field can be one of the following

AP_NONE
No ANYNET function will be supported. The field **default_preference** must take the value AP_NATIVE.

AP_ACCESS_NODE
Use nonnative (AnyNet) routing protocols only.

AP_NATIVE_THEN_NONNATIVE
This node will support ANYNET access node functions.

AP_GATEWAY
This node will start ANYNET gateway functions. This value is only valid if **node_type** AP_NETWORK_NODE.

cp_create_parms.comp_in_series
Specifies whether the use of LZ compression preceded by RLE compression is allowed:

AP_YES

AP_NO

cp_create_parms.max_ls_exception_events
Specifies the maximum number of LS_EXCEPTION entries recorded by the node. Range 0 through 200.

cp_create_parms.max_compress_lvl
The maximum compression level supported by the node.

AP_NONE
The node does not support compression.

AP_RLE_COMPRESSION
The node can support RLE compression and decompression on LU 6.2 sessions, and RLE compression and LZ9 decompression on conventional LU sessions.

AP_LZ9_COMPRESSION
The node can support LZ9 and RLE compression and decompression.

AP_LZ10_COMPRESSION
The node can support LZ10, LZ9, and RLE compression and decompression.

AP_LZ12_COMPRESSION
The node can support LZ12, LZ10, LZ9, and RLE compression and decompression.

cp_create_parms.node_spec_data_len
This field should always be set to zero.

cp_create_parms.ptf
Array for configuring and controlling future program temporary fix (PTF) operation.

cp_create_parms.ptf[0]
REQDISCONT support. Personal Communications or Communications Server normally uses REQDISCONT to deactivate limited resource host links that are no longer required by session traffic. This byte can be used to suppress Personal Communications or Communications Server's use of REQDISCONT, or to modify the settings used on REQDISCONT requests sent by Personal Communications or Communications Server.

AP_SUPPRESS_REQDISCONT
If this bit is set, Personal Communications or Communications Server does not use REQDISCONT (all other bits in this byte are ignored).

AP_OVERRIDE_REQDISCONT
If this bit is set, Personal Communications or Communications Server overrides the normal settings on REQDISCONT, based on the following two bits:

AP_REQDISCONT_TYPE
If this bit is set, Personal Communications or Communications Server specifies a type of "immediate" on REQDISCONT. Otherwise, Personal Communications or Communications Server specifies a type of "normal". (This bit is ignored if AP_OVERRIDE_REQDISCONT is not set.)

AP_REQDISCONT_RECONTACT
If this bit is set, Personal Communications or Communications Server specifies "immediate recontact" in REQDISCONT.

Otherwise,Personal Communications or Communications Server specifies “no immediate recontact”. (This bit is ignored if AP_OVERRIDE_REQDISCONT is not set.)

cp_create_parms.ptf[1]
ERP support.

Personal Communications or Communications Server normally processes an ACTPU(ERP) as an ERP (ACTPU(ERP) requests the PU-SSCP session be reset, but, unlike ACTPU(cold), does not request implicit deactivation of the subservient LU-SSCP and PLU-SLU sessions). SNA implementations can legally process ACTPU(ERP) as if it were ACTPU(cold).

AP_OVERRIDE_ERP
If this bit is set, Personal Communications or Communications Server processes all ACTPU requests as ACTPU(cold).

cp_create_parms.ptf[2]
BIS support.

Personal Communications or Communications Server normally uses the BIS protocol prior to deactivating a limited resource LU 6.2 session. This byte allows the use of BIS to be overridden.

AP_SUPPRESS_BIS
If this bit is set, Personal Communications or Communications Server does not use the BIS protocol. Limited resource LU 6.2 session are deactivated immediately using UNBIND(cleanup).

up_time
Time (in hundredths of a second) since the node was started (or restarted).

mem_size
Size of the available storage, as obtained by storage management from the underlying operating system.

mem_used
Number of bytes of storage that are currently allocated to a process.

mem_warning_threshold
Allocation threshold beyond which storage management considers the storage resources to be constrained.

mem_critical_threshold
Allocation threshold beyond which storage management considers the storage resources to be critically constrained.

nn_functions_supported
Reserved.

functions_supported
Specifies which functions are supported. This can be one or more of the following values:

AP_NEGOTIABLE_LS
AP_SEGMENT_REASSEMBLY
AP_BIND_REASSEMBLY
AP_PARALLEL_TGS
AP_CALL_IN
AP_ADAPTIVE_PACING
AP_TOPOLOGY_AWARENESS

en_functions_supported
Specifies the end-node functions supported.

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 (can only be supported when send registered names is also 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
Specifies the current DLUR release level.

nns_dlus_served_lu_reg_supp
End node only. Specifies whether the end node's network node server supports DLUS-served LU registration.

AP_NO
Registration of DLUS-served LU registration is not supported by the network node server.

AP_YES
Registration of DLUS-served LUs is supported by the network node server.

AP_UNKNOWN
The end node does not have a network node server.

NN only: This field is set to AP_NO.

fq_nn_server_name
Fully qualified, 17 byte long, name of the current network node server. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If this node is not an end node or does not have an active network node server, this field is set to null.

current_isr_sessions

The number of active ISR sessions that are currently routed through this node. If this node is not a network node, this field is set to zero.

nn_functions2
Specifies the network node functions supported.

AP_BRANCH_AWARENESS
The node is branch aware.

branch_ntwk_arch_version
Specifies the version of the branch network architecture supported or zero if the node does not support the branch network architecture.

AP_BRANCH_AWARENESS
The node is branch aware.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_PARTNER_LU

QUERY_PARTNER_LU returns information about partner LUs that have been used by a local LU.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific partner LU, or to obtain the list information in several chunks, the **plu_alias** field should be set (or the **fqplu_name** if the **plu_alias** is set to all zeros). If the **list_options** field is set to AP_FIRST_IN_LIST, both of these fields will be ignored. The **lu_name** or **lu_alias** field must always be set. The **lu_name**, if nonzero, will be used in preference to the **lu_alias**. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered by the **fqplu_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

If **plu_alias** is set to all zeros, the **fqplu_name** value will be used; otherwise, the **plu_alias** is always used and the **fqplu_name** is ignored.

The list of partner LUs returned can be filtered according to whether they currently have any active sessions. If filtering is desired, the **active_sessions** field should be set to AP_YES (otherwise this field should be set to AP_NO).

This verb returns information that is determined when at least one session is established with the partner LU.

The QUERY_PARTNER_LU_DEFINITION verb returns definition information only.

VCB Structure

```
typedef struct query_partner_lu
{
   unsigned short   opcode;             /* verb operation code            */
   unsigned char    reserv2;            /* reserved                       */
   unsigned char    format;             /* format                         */
   unsigned short   primary_rc;         /* primary return code            */
   unsigned long    secondary_rc;       /* secondary return code          */
   unsigned char    *buf_ptr;           /* pointer to buffer              */
   unsigned long    buf_size;           /* buffer size                    */
   unsigned long    total_buf_size;     /* total buffer size required     */
   unsigned short   num_entries;        /* number of entries              */
   unsigned short   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
{
   unsigned short  overlay_size;        /* size of this 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[RD_LEN];
                                      /* resource description           */
   unsigned short  act_sess_count;     /* curr 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
{
   unsigned short  overlay_size;       /* size of this 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[RD_LEN];
                                      /* resource description           */
   unsigned short  act_sess_count;     /* curr 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          */
   unsigned short  max_mc_ll_send_size; /* max send LL size for mapped   */
                                      /* conversations                  */
   unsigned char   implicit;          /* implicit or explicit entry     */
   unsigned char   security_details;   /* conversation security detail   */
   unsigned char   duplex_support;     /* full-duplex support            */
   unsigned char   preference;        /* routing preference             */
   unsigned char   reserva[16];        /* reserved                       */
} PLU_DETAIL;
```

The application supplies the following parameters:

Supplied Parameters

opcode
AP_QUERY_PARTNER_LU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of the **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros) and **plu_alias** (or **fqplu_name** if the **plu_alias** is set to all zeros) specified (see the following parameter, **lu_name** and **plu_alias**) represents an index value that is used to specify the starting point of the actual information to be returned:

AP_FIRST_IN_LIST
The **plu_alias** and **fqplu_name** fields are ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** are set to all zeros then the LU associated with the control point (the default LU) is used.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field will be used as the index value.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

active_sessions
Active session filter. Specifies whether the returned partner LUs should be filtered according to whether they currently have any active sessions (AP_YES or AP_NO).

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

plu_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

plu_summary.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_summary.fqplu_name
17-byte fully qualified network name for the partner LU. This name is 17 bytes long and is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

plu_summary.description
Resource description (as specified on DEFINE_PARTNER_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

plu_summary.act_sess_count
Total number of active sessions between the local LU and the partner LU. If the **active_sessions** filter has been set to AP_YES, then this field will always be greater than zero.

plu_summary.partner_cp_name
17-byte fully qualified network name for the control point of the partner LU. This name is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

plu_summary.partner_lu_located
Specifies whether the control point name for the partner LU has been resolved (AP_YES or AP_NO).

plu_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

plu_detail.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_detail.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

plu_detail.description
Resource description (as specified on DEFINE_PARTNER_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

plu_detail.act_sess_count
Total number of active sessions between the local LU and the partner LU. If the **active_sessions** filter has been set to AP_YES, then this field will always be greater than zero.

plu_detail.partner_cp_name
17-byte fully qualified network name for the control point of the partner LU. This name is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

plu_detail.partner_lu_located
Specifies whether the control point name for the partner LU has been resolved (AP_YES or AP_NO).

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 (AP_YES or AP_NO).

plu_detail.conv_security
Specifies whether conversation security information can be sent to this partner LU (AP_YES or AP_NO). If it is set to AP_NO, then any security information supplied by a transaction program is not sent to the partner LU. If there are currently no active sessions to this partner LU, this is set to AP_UNKNOWN.

plu_detail.max_mc_ll_send_size
Maximum size of logical length (LL) record that can be sent to the partner LU. Data records that are larger than this are broken down into several LL records before being sent to the partner LU. The maximum value **max_mc_ll_send_size** can take is 32 767.

plu_detail.implict
Specifies whether the entry is the result of an implicit (AP_YES) or explicit (AP_NO) definition.

plu_detail.security_details
Returns the conversation security support as negotiated on the BIND. This can be one or more of the following values:

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 local and partner LU agree to accept already verified requests to allocate a conversation. An already verified request need carry only a user ID, and not a password.

AP_PERSISTENT_VERIFICATION
Persistent verification is supported on the session between the local and partner LUs. This means that, once the initial request (carrying a user ID and, typically, a password) for a conversation has been verified, subsequent requests for a conversation need only carry 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.

Note: If the session does not support password substitution, then an ALLOCATE or SEND_CONVERSATION with security type of AP_PGM_STRONG will fail.

AP_UNKNOWN
There are currently no active sessions to this partner LU.

plu_detail.duplex_support
Returns the conversation duplex support as negotiated on the BIND. This is one of the following values:

AP_HALF_DUPLEX
Only half-duplex conversations are supported.

AP_FULL_DUPLEX
Full-duplex as well as half-duplex conversations are supported.

AP_UNKNOWN
The conversation duplex support is not known because there are no active sessions to the partner LU.

plu_detail.preference
Returns the routing protocols preference as specified in the DEFINE_PARTNER_LU verb.

AP_NATIVE
Use native (APPN) routing protocols only.

AP_NONNATIVE
Use nonnative (AnyNet) protocols, and if the partner LU cannot be located, then retry session activation using nonnative (AnyNet) protocols.

AP_NATIVE_THEN_NONNATIVE
Try native (APPN) protocols, and if the partner LU cannot be located then retry session activation using native (APPN) protocols.

AP_USE_DEFAULT_PREFERENCE
Use the default preference defined when the node was started. (This is set on START_NODE and can be recalled by QUERY_NODE.)

Note that nonnative routing is only meaningful when an AnyNet DLC is available to the Program, and there is an AnyNet Link Station defined. See "DEFINE_LS" on page 74 for more information.

If the field **anynet_supported** supplied on START_NODE was set to AP_NO this field must take the value AP_NATIVE or AP_USE_DEFAULT_PREFERENCE.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PLU_NAME

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_PARTNER_LU_DEFINITION

QUERY_PARTNER_LU_DEFINITION returns information that had previously been passed in on a DEFINE_PARTNER_LU verb.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific partner LU, or to obtain the list information in several chunks, the **plu_alias** field (or the **fqplu_name** if the **plu_alias** is set to all zeros) should be set. If the **plu_alias** field is nonzero it will be used to determine the index and the **fqplu_name** is ignored. If the **plu_alias** field is set to all zeros, the **fqplu_name** will be used to determine the index. If the **list_options** field is set to AP_FIRST_IN_LIST then both of these fields will be ignored. (In this case the returned list will be ordered by **plu_alias** if the AP_LIST_BY_ALIAS **list_options** is set, otherwise it will be ordered by **fqplu_name**). See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered on either **plu_alias** or **fqplu_name** according to the options specified. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering). If AP_LIST_FROM_NEXT is selected the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

Note this verb returns definition information only. The QUERY_PARTNER_LU verb returns information that is determined when at least one session is established with the partner LU.

VCB Structure

```
typedef struct query_partner_lu_definition
{
   unsigned short  opcode;             /* verb operation code          */
   unsigned char   reserv2;            /* reserved                     */
   unsigned char   format;             /* format                       */
   unsigned short  primary_rc;         /* primary return code          */
   unsigned long   secondary_rc;       /* secondary return code        */
   unsigned char   *buf_ptr;           /* pointer to buffer            */
   unsigned long   buf_size;           /* buffer size                  */
   unsigned long   total_buf_size;     /* total buffer size required   */
   unsigned short  num_entries;        /* number of entries            */
   unsigned short  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
{
   unsigned short  overlay_size;       /* size of this entry           */
   unsigned char   plu_alias[8];       /* partner LU alias             */
   unsigned char   fqplu_name[17];     /* fully qualified partner      */
                                       /* LU name                      */
   unsigned char   description[RD_LEN];
                                       /* resource description         */
} PARTNER_LU_DEF_SUMMARY;

typedef struct partner_lu_def_detail
{
   unsigned short  overlay_size;       /* size of this 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[RD_LEN]; /* resource description          */
   unsigned char   plu_un_name[8];      /* partner LU uninterpreted name */
   unsigned char   preference;          /* routing preference            */
   unsigned short  max_mc_ll_send_size;
                                        /* max MC send LL size           */
   unsigned char   conv_security_ver;   /* already_verified accepted     */
   unsigned char   parallel_sess_supp;  /* parallel sessions supported?  */
   unsigned char   reserv2[8];          /* reserved                      */
} PLU_CHARS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_PARTNER_LU_DEFINITION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **plu_alias** (or the **fqplu_name** if the **plu_alias** is set to all zeros) specified (see the following parameter, **plu_alias**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

AP_LIST_BY_ALIAS
The returned list is ordered by **plu_alias**. This option is only valid when AP_FIRST_IN_LIST is specified. If AP_LIST_FROM_NEXT or AP_LIST_INCLUSIVE is specified, the list ordering will depend on whether the **plu_alias** or **fqplu_name** has been supplied as a starting point.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**

partner_lu_def_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

partner_lu_def_summary.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

partner_lu_def_summary.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

partner_lu_def_summary.description
Resource description (as specified on DEFINE_PARTNER_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

partner_lu_def_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

partner_lu_def_detail.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

partner_lu_def_detail.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

partner_lu_def_detail.plu_chars.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

partner_lu_def_detail.plu_chars.plu_alias
Partner LU alias.

partner_lu_def_detail.plu_chars.description
Resource description (as specified on DEFINE_PARTNER_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

partner_lu_def_detail.plu_chars.plu_un_name
Uninterpreted name of the partner LU. This is an 8-byte type-A EBCDIC character string.

partner_lu_def_detail.plu_chars.preference
The set of routing protocols to be preferred for session activation to this partner LU. This field can take the following values:

AP_NATIVE
Use native (APPN) routing protocols only.

AP_NONNATIVE
Use nonnative (AnyNet) routing protocols only.

AP_NATIVE_THEN_NONNATIVE
Try native (APPN) protocols, and if the partner LU cannot be located then retry session activation using nonnative (AnyNet) protocols.

AP_NONNATIVE_THEN_NATIVE
Try nonnative (AnyNet) protocols, and if the partner LU cannot be located then retry session activation using native (APPN) protocols.

AP_USE_DEFAULT_PREFERENCE
Use the default preference defined when the node was started.

Note: Nonnative routing is only meaningful when an AnyNet DLC is available to the Node Operator Facility, and there is an AnyNet link station defined.

partner_lu_def_detail.plu_chars.max_mc_ll_send_size
Maximum size of logical length (LL) record that can be sent to the partner LU. Data records that are larger than this are broken down into several LL records before being sent to the partner LU. The maximum value **max_mc_ll_send_size** can take is 32 767.

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 can set the already verified indicator in an Attach request.

AP_YES
AP_NO

partner_lu_def_detail.plu_chars.parallel_sess_supp
Specifies whether parallel sessions are supported (AP_YES or AP_NO).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PLU_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

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).

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific port, or to obtain the list information in several chunks, the **port_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **port_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

The list of ports returned can be filtered by the name of the DLC that they belong to. In this case the **dlc_name** field should be set (otherwise this field should be set to all zeros).

VCB Structure

```
typedef struct query_port
{
   unsigned short  opcode;            /* verb operation code            */
   unsigned char   attributes;        /* Verb attributes                */
   unsigned char   format;            /* format                         */
   unsigned short  primary_rc;        /* primary return code            */
   unsigned long   secondary_rc;      /* secondary return code          */
   unsigned char   *buf_ptr;          /* pointer to buffer              */
   unsigned long   buf_size;          /* buffer size                    */
   unsigned long   total_buf_size;    /* total buffer size required     */
   unsigned short  num_entries;       /* number of entries              */
   unsigned short  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
{
   unsigned short  overlay_size;      /* size of this entry             */
   unsigned char   port_name[8];      /* port name                      */
   unsigned char   description[RD_LEN];
                                      /* resource description           */
   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
{
   unsigned short  overlay_size;      /* size of this 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                       */
   unsigned short  def_ls_good_xids;   /* number of successful XIDs      */
   unsigned short  def_ls_bad_xids;    /* number of unsuccessful XIDs    */
   unsigned short  dyn_ls_good_xids;   /* successful XIDs on dynamic     */
                                       /* LS count                       */
   unsigned short  dyn_ls_bad_xids;    /* failed XIDs on dynamic         */
   unsigned short  num_implicit_links; /* number of implicit links       */
                                       /* active on this port            */
   unsigned char   neg_ls_supp;        /* are negotiable LSs supported?  */
                                       /* LS count                       */
   unsigned char   abm_ls_supp;        /* are ABM LSs supported?         */
   unsigned long   start_time          /* start time                     */
   unsigned char   reserva[12];        /* reserved                       */
} PORT_DET_DATA;

typedef struct port_def_data
{
   unsigned char   description;        /* resource description           */
   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   implicit_uplink_to_en;
                                       /* implicit links to EN are uplink */
   unsigned char   reserv3[2];         /* NB_BYTE                        */
   unsigned long   port_number;        /* port number                    */
   unsigned short  max_rcv_btu_size;   /* max receive BTU size           */
   unsigned short  tot_link_act_lim;   /* total link activation limit    */
   unsigned short  inb_link_act_lim;   /* inbound link activation limit  */
   unsigned short  out_link_act_lim;   /* outbound link activation limit */
   unsigned char   ls_role;            /* initial link station role      */
   unsigned char   retry_flags;        /* conditions for automatic retrys */
                                       /* retries                        */
   unsigned short  max_activation_attempts;                              */
                                       /* how many automatic retries     */
   unsigned short  activation_delay_timer;
                                       /* delay between automatic retries */
   unsigned char   reserv1[10];        /* reserved                       */
   unsigned char   implicit_dspu_template[8];
                                       /* implicit DSPU template         */
   unsigned short  implicit_ls_limit   /* max number of implicit links   */
   unsigned char   reserv2             /* reserved                       */
   unsigned char   implicit_dspu_services;
                                       /* implicit links support DSPUs   */
   unsigned short  implicit_deact_timer;
                                       /* Implicit link HPR link         */
                                       /* deactivation timer             */
   unsigned short  act_xid_exchange_limit;
                                       /* activation XID exchange limit  */
   unsigned short  nonact_xid_exchange_limit;
                                       /* non-act. XID exchange limit    */
   unsigned char   ls_xmit_rcv_cap;    /* LS transmit-rcv capability     */
   unsigned char   max_ifrm_rcvd;      /* max number of I-frames that    */
                                       /* can be received                */
   unsigned short  target_pacing_count;
                                       /* target pacing count            */
   unsigned short  max_send_btu_size;  /* max send BTU size              */
   LINK_ADDRESS    dlc_data;           /* DLC data                       */
   LINK_ADDRESS    hpr_dlc_data;       /* HPR DLC data                   */
   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;
                                    /* Implicit links support HPR     */
   unsigned char   implicit_link_lvl_error;
                                    /* Implicit links support         */
                                    /* HPR link-level error recovery  */
   unsigned char   retired1;        /* reserved                       */
   TG_DEFINED_CHARS default_tg_chars;  /* Default TG chars            */
   unsigned char   discovery_supported;
                                    /* Discovery function supported?  */
   unsigned short  port_spec_data_len; /* length of port spec data    */
   unsigned short  link_spec_data_len; /* length of link spec data    */
} PORT_DEF_DATA;
typedef struct link_address
{
        unsigned short  length;        /* length                      */
        unsigned short  reserve1;      /* reserved                    */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                       /* 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;   /* modem class                 */
        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 port_spec_data
{
       unsigned char  port_data[SIZEOF_PORT_SPEC_DATA];

}  PORT_SPEC_DATA;
typedef struct link_spec_data
{
       unsigned char  link_data[SIZEOF_LINK_SPEC_DATA];

}  LINK_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_PORT

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **port_name** specified (see the following parameter, **port_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

port_name
Name of port being queried. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dlc_name
DLC name filter. This should be set to all zeros or an 8-byte string in a locally displayable character set. If this field is set then only ports belonging to this DLC are returned. This field is ignored if it is set to all zeros.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

port_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

port_summary.port_name
Name of port associated with this link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

port_summary.description
Resource description (as specified on DEFINE_PORT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

port_summary.port_state
Specifies the current state of the port.

AP_NOT_ACTIVE
AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE

port_summary.dlc_name
Name of the DLC. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

port_detail.overlay_size
The number of bytes in this entry (including any **link_spec_data**), and hence the offset to the next entry returned (if any).

port_detail.port_name
Name of port associated with this link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

port_detail.det_data.port_state
Specifies the current state of the port.

AP_NOT_ACTIVE
AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE

port_detail.det_data.dlc_type
Type of DLC. The Program supports the following types:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

port_detail.det_data.port_sim_rim
Specifies whether Set Initialization Mode (SIM) and Receive Initialization Mode (RIM) are supported (AP_YES or AP_NO).

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** field of PORT_DEF_DATA.

port_detail.det_data.neg_ls_supp
Support for negotiable link stations, AP_YES or AP_NO.

port_detail.det_data.abm_ls_supp
Support for ABM link stations. This is not known until the DLC is started

AP_NO
AP_YES
AP_UNKNOWN

port_detail.det_data.start_time
Time elapsed between the time the node was started and the last time this port was started, measured in hundredths of a second. If this port was started, zero is returned in this field.

port_detail.def_data.description
Resource description (as specified on DEFINE_PORT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

port_detail.def_data.dlc_name
Name of associated DLC. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

port_detail.def_data.port_type
Specifies the type of line used by the port. The value corresponds to one of the following values:

AP_PORT_NONSWITCHED
AP_PORT_SWITCHED
AP_PORT_SATF

port_detail.def_data.port_attributes[0]
This is the bit field. It may take the value AP_NO, or the following:

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 field **port_type** is set to AP_PORT_SWITCHED.

port_detail.def_data.implicit_uplink_to_en
BrNN only: Specifies whether implicit link stations off this port are uplink or downlink if the adjacent node is an end node. The value of this field will only be considered if there are no existing links to the same partner, as such links are used first to determine the link type.

AP_NO
Implicit links are downlink.

AP_YES
Implicit links are uplink.

Other node types: This is set to AP_NO.

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. This can be negotiable (AP_LS_NEG), primary (AP_LS_PRI), or secondary (AP_LS_SEC). Reserved if **implicit_hpr_support** is set to AP_NO.

port_detail.def_data.implicit_dspu_template
Specifies the DSPU template, defined with the DEFINE_DSPU_TEMPLATE verb, that 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 fails. This is an 8-byte string in a locally-displayable character set. All 8 bytes are significant and must be set.

If the **def_data.implicit_dspu_services** field is not set to AP_PU_CONCENTRATION, then this field is reserved.

port_detail.def_data.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. A value of 0 means that there is no limit, a value of AP_NO_IMPLICIT_LINKS means that no implicit links are allowed.

port_detail.def_data.retry_flags
This field specifies the conditions under which activation of this port are subject to automatic retry if the flag AP_INHERIT_RETRY is set on DEFINE_LS in **def_data.retry_flags**. It is a bit field, and may take any of the following values bit-wise ORed together.

AP_RETRY_ON_START
Link activation will be retried if no response is received from the remote node when activation is attempted. If the underlying port is inactive when activation is attempted, the Program will attempt to activate it.

AP_RETRY_ON_FAILURE
Link activation will be retried if the link fails while active or pending active. If the underlying port has failed when activation is attempted, the Program attempts to activate it.

AP_RETRY_ON_DISCONNECT
Link activation will be retried if the link is stopped normally by the remote node.

AP_DELAY_APPLICATION_RETRIES
Link activation retries, initiated by applications (using START_LS or on-demand link activation) will be paced using the **activation_delay_timer**.

AP_DELAY_INHERIT_RETRY
In addition to the retry conditions specified by flags in this field, those specified in the **retry_flags** field of the underlying port definition will also be used.

port_detail.def_data.max_activation_attempts
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags** and **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS.

This field specifies the number of retry attempts the Program allows when the remote node is not responding, or the underlying port is inactive. This includes both automatic retries and application-driven activation attempts.

If this limit is ever reached, no further attempts are made to automatically retry. This condition is reset by STOP_LS, STOP_PORT, STOP_DLC or a successful activation. START_LS or OPEN_LU_SSCP_SEC_RQ results in a single activation attempt, with no retry if activation fails.

Zero means 'no limit'. The value AP_USE_DEFAULTS results in the use of **max_activiation_attempts** supplied on DEFINE_PORT.

ls_detail.def_data.activation_delay_timer
This field has no effect unless at least one flag is set in DEFINE_LS in **def_data.retry_flags** and **def_data.max_activation_attempts** on DEFINE_LS is set to AP_USE_DEFAULTS.

This field specifies the number of seconds that the Program waits between automatic retry attempts, and between application-driven activation attempts if the AP_DELAY_APPLICATION_RETRIES bit is set in **def_data.retry_flags**.

The value AP_USE_DEFAULTS results in the use of **activiation_delay_timer** supplied on DEFINE_PORT.

If zero is specified, the Program uses a default timer duration of thirty seconds.

ls_detail.def_data.implicit_dspu_template
Specifies the DSPU template, defined with the DEFINE_DSPU_TEMPLATE verb, that 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 fails. This is an 8-byte string in a locally-displayable character set. All 8 bytes are significant and must be set.

If the **def_data.implicit_dspu_services** field is not set to AP_PU_CONCENTRATION, then this field is reserved.

ls_detail.def_data.implicit_dspu_services
Specifies the services that the local node will provide to the downstream PU across implicit links activated on this port. This is set to one of the following values:

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 (and will put in place definitions as specified by the DSPU template specified in the field **def_data.implicit_dspu_template**).

AP_NONE
Local node will provide no services for this downstream PU.

ls_detail.def_data.implicit_deact_timer
Limited resource link deactivation timer (in seconds). If **implicit_limited_resource** is set to AP_YES or AP_NO_SESSIONS, then an HPR-capable implicit link is automatically deactivated if no data traverses the link for the duration of this timer, and no sessions are using the link.

If **implicit_limited_resource** is set to AP_INACTIVITY then an implicit link is automatically deactivated if no data traverses the link for the duration of this timer.

If zero is specified the default value of 30 is used. Otherwise the minimum value is 5. (If it is set any lower, the specified value will be ignored and 5 will be used.) Note that this parameter is reserved unless **implicit_limited_resource** is set to AP_NO.

port_detail.def_data.act_xid_exchange_limit
Activation XID exchange limit.

port_detail.def_data.nonact_xid_exchange_limit
Nonactivation XID exchange limit.

port_detail.def_data.ls_xmit_rcv_cap
Specifies the link station transmit/receive capability. This is either two-way simultaneous (AP_LS_TWS) or two way alternating (AP_LS_TWA).

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 for BINDs on this TG. The number is only significant when fixed bind pacing is being performed. Personal Communications or Communications Server does not currently use this value.

port_detail.def_data.max_send_btu_size
Maximum BTU size that can be sent.

port_detail.def_data.dlc_data.length
Port address length.

port_detail.def_data.dlc_data.address
Port address.

port_detail.def_data.hpr_dlc_data.length
HPR Port address length.

port_detail.def_data.hpr_dlc_data.address
HPR Port address. This is currently used when supporting HPR links. The field specifies the information sent by Personal Communications or Communications Server in the X'80' subfield of the X'61' control vector on XID3 exchanged on link stations using this port.

port_detail.def_data.implicit_cp_cp_sess_support
Specifies whether CP-CP sessions are permitted for implicit link stations off this port (AP_YES or AP_NO).

port_detail.def_data.implicit_limited_resource
Specifies whether implicit link stations off this port should be deactivated when there are no sessions using the link. This is set to one of the following values:

AP_NO
Implicit links are not limited resources and will not be deactivated automatically.

AP_YES or AP_NO_SESSIONS
Implicit links are a limited resource and will be deactivated automatically when no active sessions are using them.

AP_INACTIVITY
Implicit links are a limited resource and will be deactivated automatically when no active sessions are using them, or when no data has followed on the link for the time period specified by the **implicit_deact_timer** field.

port_detail.def_data.implicit_hpr_support
Specifies whether HPR is supported on implicit links (AP_YES or AP_NO).

port_detail.def_data.implicit_link_lvl_error
Specifies whether HPR traffic is sent on implicit links using link-level error recovery (AP_YES or AP_NO).

port_detail.def_data.default_tg_chars
TG characteristics (See “DEFINE_COS” on page 35). These are used for implicit link stations off this port and also for defined link stations which specify **use_default_tg_chars**.

port_detail.def_data.discovery_supported
Specifies whether Discovery search functions are performed on this port (AP_YES or AP_NO).

port_detail.def_data.port_spec_data_len
Unpadded length, in bytes, of data passed unchanged to the port on the ACTIVATE_PORT signal. The data is concatenated to the PORT_DETAIL structure.

port_detail.def_data.link_spec_data_len
Data passed unchanged to the link station component during initialization. The data is concatenated to the PORT_DETAIL structure immediately following the port-specific data. The port-specific data and the link-specific

data together will be padded to end on a 4-byte boundary. There will be no explicit padding between the port-specific data and the link-specific data.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PORT_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_PU

QUERY_PU returns a list of local PUs and the links associated with them.

The information is returned as a list. To obtain information about a specific PU, or to obtain the list information in several chunks, the **pu_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

The verb specifies whether local PUs are attached directly to the host system or attached via DLUR. The **host_attachment** field can be used as a filter so that only information about the specified attachment type is returned.

VCB Structure

```
typedef struct query_pu
{
   unsigned short  opcode;            /* verb operation code           */
   unsigned char   attributes;        /* Verb attributes               */
   unsigned char   format;            /* format                        */
   unsigned short  primary_rc;        /* primary return code           */
   unsigned long   secondary_rc;      /* secondary return code         */
   unsigned char   *buf_ptr;          /* pointer to buffer             */
   unsigned long   buf_size;          /* buffer size                   */
   unsigned long   total_buf_size;    /* total buffer size required    */
   unsigned short  num_entries;       /* number of entries             */
   unsigned short  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               */
} QUERY_PU;
typedef struct pu_data
{
   unsigned short  overlay_size;      /* size of this entry            */
   unsigned char   pu_name[8];        /* PU name                       */
   unsigned char   description[RD_LEN];
                                      /* resource description          */
   unsigned char   ls_name[8];        /* LS name                       */
   unsigned char   pu_sscp_sess_active;
                                      /* Is PU-SSCP session active     */
   unsigned char   host_attachment;   /* Host attachment               */
   SESSION_STATS   pu_sscp_stats;     /* PU-SSCP session statistics    */
   unsigned char   sscp_id[6];        /* SSCP ID                       */
   unsigned char   conventional_lu_compression;
                                      /* Data compression requested    */
                                      /* for conventional LU sessions  */
   unsigned char   conventional_lu_cryptography;
                                      /* Cryptography required for     */
                                      /* conventional LU sessions      */
   unsigned char   reserva[12];       /* reserved                      */
} PU_DATA;
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; /* max send BTU size             */
   unsigned short  max_rcv_btu_size;  /* max rcv BTU size              */
   unsigned short  max_send_pac_win;  /* max send pacing window size   */
   unsigned short  cur_send_pac_win;  /* curr send pacing window size  */
   unsigned short  max_rcv_pac_win;   /* max recv 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 of FMD data frames sent   */
   unsigned long   send_data_bytes;   /* number of data bytes sent     */
   unsigned long   rcv_data_frames;   /* num data frames received      */
   unsigned long   rcv_fmd_data_frames;
                                 /* num of FMD data frames rcvd   */
   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   pacing_type;       /* type of pacing in use         */
} SESSION_STATS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_PU

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

The **pu_name** specified (see the following parameter, **pu_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

pu_name
Name of the first PU to be listed. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

host_attachment
Filter for host attachment:

AP_NONE
Return information about all local PUs.

AP_DLUR_ATTACHED
Return information about all local PUs that are supported by DLUR.

AP_DIRECT_ATTACHED
Return information about only those PUs that are directly attached to the host system.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

pu_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

pu_data.pu_name
PU name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

pu_data.description
Resource description (as specified on DEFINE_LS or DEFINE_INTERNAL_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

pu_data.ls_name
Name of the link station associated with this PU. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

pu_data.pu_sscp_sess_active
Specifies whether the PU-SSCP session is active (AP_YES or AP_NO).

pu_data.host_attachment
Local PU host attachment type:

AP_DLUR_ATTACHED
PU is attached to host system using DLUR.

AP_DIRECT_ATTACHED
PU is directly attached to host system.

pu_data.pu_sscp_stats.rcv_ru_size
This field is always reserved.

pu_data.pu_sscp_stats.send_ru_size
This field is always reserved.

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
This field will always be set to zero.

pu_data.pu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

pu_data.pu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

pu_data.pu_sscp_stats.cur_rcv_pac_win
This field will always be 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.

pu_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 address indicator. When bringing up a session, the sender of the ACTPU sets this field to zero if the local node contains the primary link station, and sets it to one if the ACTPU 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 string in a locally displayable character set. All 8 bytes are significant.

pu_data.pu_sscp_stats.pacing_type
Receiving pacing type in use on the PU-SSCP session. This will take the value AP_NONE.

pu_data.sscp_id
This is a 6–byte field containing the SSCP ID received in the ACTPU for the PU.

If **pu_sscp_sess_active** is not AP_YES, then this field will be zeroed.

pu_data.conventional_lu_compression
Specifies whether data compression is requested for sessions using this PU.

AP_NO
The local node should not be compressing or decompressing data flowing on sessions using this PU.

AP_YES
Data compression should be enabled for sessions dependent on this PU if the host requests compression.

pu_data.conventional_lu_cryptography
Specifies whether session level encryption is required for conventional LU sessions dependent on this PU.

AP_NONE
Session level encryption is not performed by the Program.

AP_MANDATORY
Mandatory session level encryption is performed by the Program if an import key is available to the LU. Otherwise, it must be performed by the application that uses the LU (if this is PU Concentration, then it is performed by a downstream LU).

AP_OPTIONAL
This value allows the cryptography used to be driven by the host application on a per session basis. If the host requests cryptography for a session dependent on this PU, then the behaviour of the Program is as for AP_MANDATORY. If the host does not request cryptography, then the behaviour is the same as AP_NONE.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_PU_TYPE
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_RTP_CONNECTION

QUERY_RTP_CONNECTION is used at a network node or an end node and returns list information about Rapid Transport Protocol (RTP) connections for which the node is an endpoint.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific RTP connection, or to obtain the list information in several chunks, the **rtp_name** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **rtp_name**. Ordering is according to name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with normal MIB ordering). If AP_LIST_FROM_NEXT is selected the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct query_rtp_connection
{
   unsigned short  opcode;            /* verb operation code           */
   unsigned char   reserv2;           /* reserved                      */
   unsigned char   format;            /* format                        */
   unsigned short  primary_rc;        /* Primary return code           */
   unsigned long   secondary_rc;      /* Secondary return code         */
   unsigned char   *buf_ptr;          /* pointer to buffer             */
   unsigned long   buf_size;          /* buffer size                   */
   unsigned long   total_buf_size;    /* total buffer size required    */
   unsigned short  num_entries;       /* number of entries             */
   unsigned short  total_num_entries; /* total number of entries       */
   unsigned char   list_options;      /* listing options               */
   unsigned char   reserv3;           /* reserved                      */
   unsigned char   rtp_name[8];       /* name of RTP connection        */
} QUERY_RTP_CONNECTION;
```

```
typedef struct rtp_connection_summary
{
   unsigned short  overlay_size;      /* size of this entry            */
   unsigned char   rtp_name[8];       /* RTP connection name           */
   unsigned char   first_hop_ls_name[8];
                                      /* LS name of first hop          */
   unsigned char   dest_node_name[17]; /* fully qualified name of      */
                                      /* destination node              */
   unsigned char   reserv1;           /* reserved                      */
   unsigned char   cos_name[8];       /* class-of-service name         */
   unsigned short  num_sess_active;   /* number of active sessions     */
} RTP_CONNECTION_SUMMARY;
```

```
typedef struct rtp_connection_detail
{
   unsigned short  overlay_size;      /* size of this entry            */
   unsigned char   rtp_name[8];       /* RTP connection name           */
   unsigned char   first_hop_ls_name[8];
                                      /* LS name of first hop          */
   unsigned char   dest_node_name[17]; /* fully qualified name of      */
                                      /* destination node              */
   unsigned char   isr_boundary_fn;   /* connection provides ISR BF    */
   unsigned char   reserv1[3];        /* reserved                      */
   unsigned char   cos_name[8];       /* class-of-service name         */
   unsigned short  max_btu_size;      /* max BTU size                  */
   unsigned long   liveness_timer;    /* liveness timer                */
   unsigned char   local_tcid[8];      /* local TCID                    */
   unsigned char   remote_tcid[8];     /* remote TCID                   */
   RTP_STATISTICS  rtp_stats;          /* RTP statistics                */
   unsigned short  num_sess_active;    /* number of active sessions     */
   unsigned char   reserv2[16];        /* reserved                      */
   unsigned short  rscv_len;           /* length of appended RSCV       */
} RTP_CONNECTION_DETAIL;
```

```
typedef struct rtp_statistics
{
   unsigned long   bytes_sent;         /* total number of bytes sent    */
   unsigned long   bytes_received;     /* total number of bytes received */
   unsigned long   bytes_resent;       /* total number of bytes resent  */
   unsigned long   bytes_discarded;    /* total number bytes discarded  */
   unsigned long   packets_sent;       /* total number of packets sent  */
   unsigned long   packets_received;   /* total number packets received */
   unsigned long   packets_resent;     /* total number of packets resent */
   unsigned long   packets_discarded;  /* total number packets discarded */
   unsigned long   gaps_detected;      /* gaps detected                 */
   unsigned long   send_rate;          /* current send rate             */
   unsigned long   max_send_rate;      /* maximum send rate             */
   unsigned long   min_send_rate;      /* minimum send rate             */
   unsigned long   receive_rate;       /* current receive rate          */
   unsigned long   max_receive_rate;   /* maximum receive rate          */
   unsigned long   min_receive_rate;   /* minimum receive rate          */
   unsigned long   burst_size;         /* current burst size            */
   unsigned long   up_time;            /* total uptime of connection    */
   unsigned long   smooth_rtt;         /* smoothed round-trip time      */
   unsigned long   last_rtt;           /* last round-trip time          */
   unsigned long   short_req_timer;    /* SHORT_REQ timer duration      */
   unsigned long   short_req_timeouts; /* number of SHORT_REQ timeouts  */
   unsigned long   liveness_timeouts;  /* number of liveness timeouts   */
   unsigned long   in_invalid_sna_frames;
                                       /* number of invalid SNA frames  */
                                       /* received                      */
   unsigned long   in_sc_frames;       /* number of SC frames received  */
   unsigned long   out_sc_frames;      /* number of SC frames sent      */
   unsigned char   reserve[40];        /* reserved                      */
} RTP_STATISTICS;
```

Note: The **rtp_connection_detail** overlay will be followed by a Route Selection Control Vector (RSCV) as defined by SNA. After RTP connection setup and before any path switch, the RSCV for an RTP connection will be stored and displayed at each node as follows:

- The RSCV will contain all the hops from the local node to the partner RTP node.
- If the partner RTP node is not an endpoint of the session causing the RTP connection to be activated, the RSCV will also store one “boundary function hop” leading away from the partner RTP node.
- The RSCV will never contain a boundary function hop leading into the local node, even if the local node does not contain a session endpoint.

After path switch has occurred, the RSCVs stored and displayed will only include the hops from the local node to the partner RTP node. (Never a boundary function hop.)

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_RTP_CONNECTION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **rtp_name** represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The **rtp_name** is ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

rtp_name
RTP connection name. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

rtp_connection_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

rtp_connection_summary.rtp_name
RTP connection name. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

rtp_connection_summary.first_hop_ls_name
Link station name of the first hop of the RTP connection. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

rtp_connection_summary.dest_node_name
Fully qualified, 17-byte name of the destination node of the RTP connection composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

rtp_connection_summary.cos_name
Class-of-service name for the RTP connection. This is an 8-byte alphanumeric type-A EBCDIC character string, padded to the right with EBCDIC spaces.

rtp_connection_summary.num_sess_active
Number of sessions currently active on the RTP connection.

rtp_connection_detail.overlay_size
The number of bytes in this entry (including any appended RSCV), and hence the offset to the next entry returned (if any).

rtp_connection_detail.rtp_name
RTP connection name. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

rtp_connection_detail.first_hop_ls_name
Link station name of the first hop of the RTP connection. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

rtp_connection_detail.dest_node_name
Fully qualified, 17-byte name of the destination node of the RTP connection composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded-with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

rtp_connection_detail.isr_boundary_fn
AP_YES if the RTP Connection is being used for any ISR session for which the local node is providing HPT-APPN Boundary Function, AP_NO otherwise.

rtp_connection_detail.cos_name
Class-of-service name for the RTP connection. This is an 8-byte alphanumeric type-A EBCDIC character string, padded to the right with EBCDIC spaces.

rtp_connection_detail.max_btu_size
Maximum BTU size for the RTP connection measured in bytes.

rtp_connection_detail.liveness_timer
Liveness timer for the RTP connection, measured in seconds.

rtp_connection_detail.local_tcid
Local TCID for the RTP connection.

rtp_connection_detail.remote_tcid
Remote TCID for the RTP connection.

rtp_connection_detail.rtp_stats.bytes_sent
Total number of bytes that the local node has sent on this RTP connection.

rtp_connection_detail.rtp_stats.bytes_received
Total number of bytes that the local node has received on this RTP connection.

rtp_connection_detail.rtp_stats.bytes_resent
Total number of bytes resent by the local node owing to loss in transit.

rtp_connection_detail.rtp_stats.bytes_discarded
Total number of bytes sent by the other end of the RTP connection and were discarded as duplicates of data already received.

rtp_connection_detail.rtp_stats.packets_sent
Total number of packets that the local node has sent on this RTP connection.

rtp_connection_detail.rtp_stats.packets_received
Total number of packets that the local node has received on this RTP connection.

rtp_connection_detail.rtp_stats.packets_resent
Total number of packets resent by the local node owing to loss in transit.

rtp_connection_detail.rtp_stats.packets_discarded
Total number of packets sent by the other end of the RTP connection that were discarded as duplicates of data already received.

rtp_connection_detail.rtp_stats.gaps_detected
Total number of gaps detected by the local node. Each gap corresponds to one or more lost frames.

rtp_connection_detail.rtp_stats.send_rate
Current send rate on this RTP connection (measured in kilobits per second). This is the maximum allowed send rate as calculated by the ARB algorithm.

rtp_connection_detail.rtp_stats.max_send_rate
Maximum send rate on this RTP connection (measured in kilobits per second).

rtp_connection_detail.rtp_stats.min_send_rate
Minimum send rate on this RTP connection (measured in kilobits per second).

rtp_connection_detail.rtp_stats.receive_rate
Current receive rate on this RTP connection (measured in kilobits per second). This is the actual receive rate calculated over the last measurement interval.

rtp_connection_detail.rtp_stats.max_receive_rate
Maximum receive rate on this RTP connection (measured in kilobits per second).

rtp_connection_detail.rtp_stats.min_receive_rate
Minimum receive rate on this RTP connection (measured in kilobits per second).

rtp_connection_detail.rtp_stats.burst_size
Current burst size on the RTP Connection measured in bytes.

rtp_connection_detail.rtp_stats.up_time
Total number of seconds the RTP connection has been active.

rtp_connection_detail.rtp_stats.smooth_rtt
Smoothed measure of round-trip time between the local node and the partner RTP node (measured in milliseconds).

rtp_connection_detail.rtp_stats.last_rtt
The last measured round-trip time between the local node and the partner RTP node (measured in milliseconds).

rtp_connection_detail.rtp_stats.short_req_timer
The current duration used for the SHORT_REQ timer (measured in milliseconds).

rtp_connection_detail.rtp_stats.short_req_timeouts
Total number of times the SHORT_REQ timer has expired for this RTP connection.

rtp_connection_detail.rtp_stats.liveness_timeouts
Total number of times the liveness timer has expired for this RTP connection. The liveness timer expires when the connection has been idle for the period specified in **rtp_connection_detail.liveness_timer**.

rtp_connection_detail.rtp_stats.in_invalid_sna_frames
Total number of SNA frames received and discarded as not valid on this RTP connection.

rtp_connection_detail.rtp_stats.in_sc_frames
Total number of session control frames received on this RTP connection.

rtp_connection_detail.rtp_stats.out_sc_frames
Total number of session control frames sent on this RTP connection.

rtp_connection_detail.num_sess_active
Number of sessions currently active on the RTP connection.

rtp_connection_detail.rscv_len
Length of the appended Route Selection Control Vector for the RTP connection. (If none is appended, then the length is zero.) The RSCV will be padded to end on a 4-byte boundary to ensure correct alignment of the next detail entry, but the **rscv_len** does not include this padding.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_RTP_CONNECTION

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_SESSION

QUERY_SESSION returns list information about sessions for which the node is an endpoint.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific session, or to obtain the list information in several chunks, the **session_id** field should be set. Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. Note that the **lu_name** (or **lu_alias**) and **plu_alias** (or **fqplu_name**) fields must always be set. The **lu_name**, if nonzero, will be used in preference to the **lu_alias**. See “Querying the Node” on page 10, for background on how the list formats are used.

The list of sessions returned may be filtered by the name of the partner LU. To do this, the **fqplu_name** or **plu_alias** fields should be set. If **plu_alias** is set to all zeros, the **fqplu_name** value will be used, otherwise the **plu_alias** is always used and the **fqplu_name** is ignored.

The list of sessions returned can be filtered by the name of the mode that they are associated with. In this case the **mode_name** field should be set (otherwise this field should be set to all zeros).

In addition to the detail information for each session, a route selection control vector (RSCV) will be returned (in key-length format) if this is specified on the START NODE parameters. This RSCV (Specified in *SNA Formats*) defines the route through the network that the session takes in hop-by-hop form.

VCB Structure

```
typedef struct query_session
{
   unsigned short  opcode;            /* verb operation code         */
   unsigned char   reserv2;           /* reserved                    */
   unsigned char   format;            /* format                      */
   unsigned short  primary_rc;        /* primary return code         */
   unsigned long   secondary_rc;      /* secondary return code       */
   unsigned char   *buf_ptr;          /* pointer to buffer           */
   unsigned long   buf_size;          /* buffer size                 */
   unsigned long   total_buf_size;    /* total buffer size required  */
   unsigned short  num_entries;       /* number of entries           */
   unsigned short  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
{
   unsigned short  overlay_size;      /* size of this entry          */
   unsigned char   plu_alias[8];      /* partner LU alias            */
   unsigned char   fqplu_name[17];    /* fully qualified partner     */
                                      /* LU name                     */
   unsigned char   reserv3[1];        /* 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
{
   unsigned short  overlay_size;      /* size of this entry            */
   unsigned char   plu_alias[8];      /* partner LU alias              */
   unsigned char   fqplu_name[17];    /* fully qualified partner       */
                                      /* LU name                       */
   unsigned char   reserv3[1];        /* 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[6];        /* SSCP ID of host               */
   unsigned char   reserva[20];       /* reserved                      */
   unsigned long   session_start_time;/* start time of the session     */
   unsigned short  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;
typedef struct fqpcid
{
   unsigned char   pcid[8];           /* pro correlator identifier     */
   unsigned char   fqcp_name[17];     /* orig's network qualified      */
                                      /* CP name                       */
   unsigned char   reserve3[3];       /* reserved                      */
} FQPCID;
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;  /* Max send pacing window size   */
   unsigned short  cur_send_pac_win;  /* Curr send pacing window size */
   unsigned short  max_rcv_pac_win;   /* Max receive pacing win size   */
   unsigned short  cur_rcv_pac_win;   /* Curr rec pacing window size   */
   unsigned long   send_data_frames;  /* Number of data frames sent    */
   unsigned long   send_fmd_data_frames;
                                      /* num of FMD data frames sent   */
   unsigned long   send_data_bytes;   /* Number of data bytes sent     */
   unsigned long   rcv_data_frames;   /* Num data frames received      */
   unsigned long   rcv_fmd_data_frames;
                                      /* num of FMD data frames recvd */
   unsigned long   rcv_data_bytes;    /* Num 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[8];        /* Link station name             */
   unsigned char   pacing_type;       /* type of pacing in use         */
} SESSION_STATS;
```

Note: The session detail overlay 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. The RSCV is included (in key-length format) if the field on the START_NODE verb is set to AP_YES. If the START_NODE **rscv_len** is set to zero.

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_SESSION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The combination of **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros), **pu_alias** (or **fqplu_name** if the **plu_alias** is set to all zeros), **mode_name** and **session_id** specified (see the following parameters, **lu_name**, **pu_alias**, **mode_name**, and **session_id**) represent an index value that is used to specify the starting point of the actual information to be returned .

AP_FIRST_IN_LIST
The **session_id** is ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable

character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** fields are set to all zeros, the LU associated with the control point (the default LU) is used.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field will be used for determining the index.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_name
Mode name filter. This should be set to all zeros or an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set then only sessions associated with this mode are returned. This field is ignored if it is set to all zeros.

session_id
8-byte identifier of the session. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

session_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

session_summary.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

session_summary.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces.

session_summary.mode_name
Mode name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right 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 control point name. This 17-byte name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

session_detail.overlay_size
The number of bytes in this entry (including any appended RSCV), and hence the offset to the next entry returned (if any).

session_detail.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

session_detail.fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces.

session_detail.mode_name
Mode name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right 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 17-byte name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

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

session_detail.trans_pri
Transmission priority. This is set to one of the following values:

AP_LOW
AP_MEDIUM
AP_HIGH
AP_NETWORK

session_detail.ltd_res
Specifies whether the session uses a limited resource link (AP_YES or AP_NO).

session_detail.polarity
Specifies the polarity of the session (AP_PRIMARY or AP_SECONDARY).

session_detail.contention
Specifies the session contention polarity. This indicates whether the local LU has 'first refusal' for the use of this session (AP_CONWINNER) or whether it must bid before using the session (AP_CONLOSER).

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 address 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 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 data flows.

session_detail.sess_stats.pacing_type
Type of receive pacing in use on this session. This may take the values AP_NONE, AP_PACING_FIXED, or AP_PACING_ADAPTIVE.

session_detail.duplex_support
Returns the conversation duplex support as negotiated on the BIND. This is one of the following values:

AP_HALF_DUPLEX
Only half-duplex conversations are supported.

AP_FULL_DUPLEX
Full-duplex as well as half-duplex conversations are supported. Expedited data is also supported.

session_detail.sscp_id
For dependent LU sessions, this field contains the SSCP ID received in the ACTPU from the host for the PU that the local LU is mapped to. For independent LU sessions, this field will be set to all binary zeros.

session_detail.session_start_time
The time that elapsed 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, zero is returned in this field.

session_detail.session_timeout
Specifies the timeout associated with the session. This is derived from the following values:

LU6.2 timeout associated with the local LU
LU6.2 timeout associated with the remote LU
mode timeout
global timeout
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 PLU to the SLU.

AP_NONE
Compression is not used.

AP_RLE_COMPRESSION
RLE compression is used.

AP_LZ9_COMPRESSION
This node can supports LZ9 compression.

AP_LZ10_COMPRESSION
The node can supports LZ10 compression.

AP_LZ12_COMPRESSION
The node can supports LZ12 compression.

session_detail.slu_plu_comp_lvl
Specifies the compression level for data sent from the SLU to the PLU.

AP_NONE
Compression is not used.

AP_RLE_COMPRESSION
RLE compression is used.

AP_LZ9_COMPRESSION
This node can supports LZ9 compression.

AP_LZ10_COMPRESSION
The node can supports LZ10 compression.

AP_LZ12_COMPRESSION
The node can supports LZ12compression.

session_detail.rscv_len
Length of the RSCV that is appended to the **session_detail** structure. (If none is appended, then the length is zero.) The RSCV will be padded to end on a 4-byte boundary.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_SESSION_ID

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_SIGNED_ON_LIST

QUERY_SIGNED_ON_LIST retrieves information about users signed on to a particular LU.

The local LU is specified by **lu_name** or **lu_alias**. **Buf_ptr, buf_size, total_buf_size, num_entries, total_num_entries** and **overlay_size** have the usual meanings for a QUERY verb.

Entries are returned as a list of SIGNED_ON_LIST_ENTRY structures, pointed to by **buf_ptr**, or appended to the QUERY_SIGNED_ON_LIST VCB if **buf_ptr** is NULL. The list is ordered by **plu_alias/fqplu_name**, then by **user_id** and then by **profile**. If **plu_alias** is specified, **fqplu_name** is ignored.

The **list_options** can take the values AP_FIRST_IN_LIST, AP_LIST_FROM_NEXT, AP_LIST_INCLUSIVE. If **list_options** is AP_FIRST_IN_LIST, **plu_alias, fqplu_name, user_id**, and **profile** are ignored. The **list** specifies which list to return entries from, which must be AP_SIGNED_ON_TO_LIST.

VCB Structure

```
typedef struct query_signed_on_list
{
   unsigned short  opcode;            /* verb operation code         */
   unsigned char   reserv2;           /* reserved                    */
   unsigned char   format;            /* format                      */
   unsigned short  primary_rc;        /* primary return code         */
   unsigned long   secondary_rc;      /* secondary return code       */
   unsigned char   *buf_ptr;          /* pointer to buffer           */
   unsigned long   buf_size;          /* buffer size                 */
   unsigned long   total_buf_size;    /* total buffer size required  */
   unsigned short  num_entries;       /* number of entries           */
   unsigned short  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   user_id[10];       /* User ID                     */
   unsigned char   profile[10];       /* Profile                     */
   unsigned char   list;              /* Signed-on list type         */
} QUERY_SIGNED_ON_LIST;
typedef struct signed_on_list_entry
{
   unsigned short  overlay_size;      /* size of this entry          */
   unsigned char   plu_alias[8];      /* partner LU alias            */
   unsigned char   user_id[10];       /* fully qualified partner     */
   unsigned char   profile[10];       /* profile                     */
} SIGNED_ON_LIST_ENTRY;
```

Supplied Parameters

The application supplies the following parameters:

opcode
: AP_QUERY_SIGNED_ON_LIST

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information.

The combination of **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros), **pu_alias** (or **fqplu_name** if the **plu_alias** is set to all zeros), **user_id** and **profile** specified (see the following parameters, **lu_name**, **pu_alias**, **user_id**, and **profile**) represent an index value that is used to specify the starting point of the actual information to be returned .

AP_FIRST_IN_LIST
The **pu_alias, fqplu_name, user_id**, and **profile** are ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** fields are set to all zeros, the LU associated with the control point (the default LU) is used.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field will be used for determining the index.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

user_id
User ID. This should be set to a 10–byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. If this field is set then only sessions associated with this mode are returned. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

profile
This is a 10-byte alphanumeric EBCDIC string. Note, the Program currently supports only the blank profile (10 eBCDIC spaces). This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

list Signed–on list type. This must be set to AP_SIGNED_ON_TO_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

signed_on_list_entry.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

signed_on_list_entry.plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

signed_on_list_entry.user_id
User ID. This is a 10-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

signed_on_list_entry.profile
10-byte alphanumeric EBCDIC string.

Note: The Program currently supports only the blank profile (10 EBCDIC spaces).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_ALIAS

AP_INVALID_LU_NAME
AP_INVALID_PLU_NAME
AP_INVALID_USERID
AP_INVALID_PROFILE
AP_INVALID_LIST
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node stopped, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

QUERY_STATISTICS

QUERY_STATISTICS queries link station and port statistics. Personal Communications or Communications Server passes this query directly to the DLC. The format of the statistics depends on the DLC implementation.

VCB Structure

```
typedef struct query_statistics
{
   unsigned short  opcode;          /* verb operation code       */
   unsigned char   reserv2;         /* reserved                  */
   unsigned char   format;          /* format                    */
   unsigned short  primary_rc;      /* primary return code       */
   unsigned long   secondary_rc;    /* secondary return code     */
   unsigned char   name[8];         /* LS 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;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_STATISTICS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

name Name defined for the link station or port (depending on setting of **stats_type** parameter). This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. Personal Communications or Communications Server uses this to correlate the response to the correct link station or port.

stats_type
The type of resource for which statistics are requested. This must be set to one of the following values:

AP_LS
AP_PORT

table_type
The type of statistics table requested. This must be set to one of the following categories of information:

AP_STATS_TBL
Specifies that statistical information will be returned.

AP_ADMIN_TBL
Specifies that administrative information will be returned.

AP_OPER_TBL
Specifies that operational information will be returned. The format of the information returned for each category is DLC implementation specific.

reset_stats
Specifies whether the statistics should be reset (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

dlc_type
Type of the DLC. The value of this field is DLC implementation specific. The values are as follows:

AP_ANYNET
AP_LLC2
AP_OEM_DLC
AP_SDLC
AP_TWINAX
AP_X25

statistics
Current statistics of link station or port.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LINK_NAME

AP_INVALID_PORT_NAME
AP_INVALID_STATS_TYPE
AP_INVALID_TABLE_TYPE

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_LINK_DEACTIVATED

AP_PORT_DEACTIVATED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_TP

QUERY_TP returns information about transaction programs currently being used by a local LU.

The information is returned as a list. To obtain information about a specific transaction program, or to obtain the list information in several chunks, the **tp_name** field should be set. If the **list_options** field is set to AP_FIRST_IN_LIST then this field will be ignored. Note that the **lu_name** or **lu_alias** field must always be set. The **lu_name** field, if nonzero, will be used in preference to the **lu_alias** field. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is ordered by the **tp_name** using EBCDIC lexicographical ordering for names of the same length. This verb returns information that is determined once the TP starts to be used by a local LU. The QUERY_TP_DEFINITION verb returns definition information only.

VCB Structure

```
typedef struct query_tp
{
   unsigned short  opcode;             /* Verb operation code          */
   unsigned char   attributes;         /* verb attributes              */
   unsigned char   format;             /* format                       */
   unsigned short  primary_rc;         /* Primary return code          */
   unsigned long   secondary_rc;       /* Secondary return code        */
   unsigned char   *buf_ptr;           /* pointer to buffer            */
   unsigned long   buf_size;           /* buffer size                  */
   unsigned long   total_buf_size;     /* total buffer size required   */
   unsigned short  num_entries;        /* number of entries            */
   unsigned short  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
{
   unsigned short  overlay_size;       /* size of this entry           */
   unsigned char   tp_name[64];        /* TP name                      */
   unsigned char   description[RD_LEN];
                                       /* resource description         */
   unsigned short  instance_limit;     /* max instance count           */
   unsigned short  instance_count;     /* current instance count       */
   unsigned short  locally_started_count;
                                       /* locally started instance     */
                                       /* count                        */
   unsigned short  remotely_started_count;
                                       /* remotely started instance    */
                                       /* count                        */
   unsigned char   reserva[20];        /* reserved                     */
} TP_DATA;
typedef struct tp_spec_data
{
        unsigned char pathname[256];   /* path and TP name             */
        unsigned char parameters[64];  /* parameters for TP            */
        unsigned char queued;          /* queued TP (AP_YES)           */
        unsigned char load_type;        /* type of load-DETACHED/CONSOLE */
        unsigned char dynamic_load;     /* dynamic loading of TP enabled */
        unsigned char reserved[5];      /* max size is 120 bytes         */
} TP_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_TP

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information: The combination of **lu_name** (or **lu_alias** if the **lu_name** is set to all zeros) and **tp_name** specified (see the following parameters, **lu_name** and **tp_name**) represents an index value that is used to specify the starting point of the actual information to be returned .

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the

lu_name and the **lu_alias** are set to all zeros, the LU that is associated with the control point (the default LU) is used.

tp_name
Transaction program name. This is a 64-byte string, padded to the right with spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

tp_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

tp_data.tp_name
Transaction program name. This is a 64-byte string, padded to the right with spaces.

tp_data.description
Resource description (as specified on DEFINE_TP). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

tp_data.instance_limit
Maximum number of concurrently active instances of the specified transaction program.

tp_data.instance_count
Number of instances of the specified transaction program that are currently active.

tp_data.locally_started_count
Number of instances of the specified transaction program which have been started locally (by the transaction program issuing a TP_STARTED verb).

tp_data.remotely_started_count
Number of instances of the specified transaction program that have been started remotely (by a received Attach request).

tp_spec_data.pathname
Specifies the path and transaction program name.

tp_spec_data.parameters
Specifies the parameters for the transaction program.

tp_spec_data.queued
Specifies whether the transaction program will be queued.

tp_spec_data.load_type
Specifies how the transaction program will be loaded.

tp_spec_data.dynamic_load
Specifies whether the TP can be dynamically loaded (AP_YES or AP_NO).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TP_NAME

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_TP_DEFINITION

QUERY_TP_DEFINITION returns both information previously passed in on a DEFINE_TP verb and information about Personal Communications or Communications Server defined transaction programs.

The information is returned as a list in one of two formats, either summary or detailed information. To obtain information about a specific transaction program, or to obtain the list information in several chunks, the **tp_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered by the **tp_name**, using EBCDIC lexicographical ordering. If AP_LIST_FROM_NEXT is selected the returned list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

This verb returns definition information only. The QUERY_TP verb returns information that is determined once the transaction program starts to be used by a local LU.

VCB Structure

```
typedef struct query_tp_definition
{
   unsigned short  opcode;            /* Verb operation code          */
   unsigned char   attributes;        /* verb attributes              */
   unsigned char   format;            /* format                       */
   unsigned short  primary_rc;        /* Primary return code          */
   unsigned long   secondary_rc;      /* Secondary return code        */
   unsigned char   *buf_ptr;          /* pointer to buffer            */
   unsigned long   buf_size;          /* buffer size                  */
   unsigned long   total_buf_size;    /* total buffer size required   */
   unsigned short  num_entries;       /* number of entries            */
   unsigned short  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
{
   unsigned short  overlay_size;      /* size of this entry           */
   unsigned char   tp_name[64];       /* TP name                      */
   unsigned char   description[RD_LEN];
                                      /* resource description         */
} TP_DEF_SUMMARY;
typedef struct tp_def_detail
{
   unsigned short  overlay_size;      /* size of this entry           */
   unsigned char   tp_name[64];       /* TP name                      */
   TP_CHARS        tp_chars;          /* TP characteristics           */
} TP_DEF_DETAIL;
typedef struct tp_chars
{
   unsigned char   description[RD_LEN];
                                      /* resource description         */
   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 supported              */
   unsigned char   reserv3[9];        /* reserved                      */
   unsigned short  tp__instance_limit; /* limit on currently active TP  */
                                      /* instances                     */
   unsigned short  incoming_alloc_timeout;
                                      /* incoming allocation timeout   */
   unsigned short  rcv_alloc_timeout; /* receive allocation timeout    */
   unsigned short  tp_data_len;       /* TP data length                */
   TP_SPEC_DATA    tp_data;           /* TP data                       */
} TP_CHARS;

typedef struct tp_spec_data
{
  unsigned char pathname[256];        /* path and TP name              */
  unsigned char parameters[64];       /* parameters for TP             */
  unsigned char queued;               /* queued TP (AP_YES)            */
  unsigned char load_type;            /* type of load-DETACHED/CONSOLE */
  unsigned char dynamic_load;         /* dynamic loading of TP enabled */
  unsigned char reserved[5];          /* max size is 120 bytes         */
} TP_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_TP_DEFINITION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information:

AP_SUMMARY
Returns summary information only.

AP_DETAIL
Returns detailed information.

The **tp_name** specified (see the following parameter, **tp_name**) represents an index value that is used to specify the starting point of the actual information to be returned:

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

tp_name
Name of the defined transaction program. This is a 64-byte string, padded to the right with spaces. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

tp_def_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

tp_def_summary.tp_name
Defined transaction program name. This is a 64-byte string, padded to the right with spaces.

tp_def_summary.description
Resource description (as specified on DEFINE_TP). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

tp_def_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

tp_def_detail.tp_name
Defined transaction program name. This is a 64-byte string, padded to the right with spaces.

tp_def_detail.tp_chars.description
Resource description (as specified on DEFINE_TP). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

tp_def_detail.tp_chars.conv_type
Specifies the types of conversation supported by the transaction program:

AP_BASIC
AP_MAPPED
AP_EITHER

tp_def_detail.tp_chars.security_rqd
Specifies whether conversation security information is required to start the transaction program (AP_NO or AP_YES).

tp_def_detail.tp_chars.sync_level
Specifies the synchronization levels supported by the transaction program:

AP_NONE
The transaction program supports a synchronization level of None.

AP_CONFIRM_SYNC_LEVEL
The transaction program supports a synchronization level of Confirm.

AP_EITHER
The transaction program supports a synchronization level of None or Confirm.

AP_SYNCPT_REQUIRED
The transaction program supports a synchronization level of Sync-point.

AP_SYNCPT_NEGOTIABLE
The transaction program supports a synchronization level of None, Confirm, or Sync-point.

tp_def_detail.tp_chars.dynamic_load
Specifies whether the transaction program can be dynamically loaded (AP_YES or AP_NO).

tp_def_detail.tp_chars.enabled
Specifies whether the transaction program can be attached successfully (AP_YES or AP_NO). The default is AP_NO.

tp_def_detail.tp_chars.pip_allowed
Specifies whether the transaction program can receive program initialization (PIP) parameters (AP_YES or AP_NO).

tp_def_detail.tp_chars.duplex_support
Indicates whether the transaction program is full or half duplex.

AP_FULL_DUPLEX
Specifies the transaction program is full duplex.

AP_HALF_DUPLEX
Specifies the transaction program is half duplex.

AP_EITHER_DUPLEX
Specifies the transaction program can be either half or full duplex

tp_def_detail.tp_chars.tp_instance_limit
Limit on the number of concurrently active transaction program instances.

tp_def_detail.tp_chars.incoming_alloc_timeout
Specifies the number of seconds that an incoming Attach will be queued waiting for a RECEIVE_ALLOCATE. Zero implies no timeout, and so it will be held indefinitely.

tp_def_detail.tp_chars.rcv_alloc_timeout
Specifies the number of seconds that a RECEIVE_ALLOCATE verb will be queued while waiting for an Attach. Zero implies no timeout, and so it will be held indefinitely.

tp_def_detail.tp_chars.tp_data_len
Length of the implementation-dependent transaction program data.

tp_def_detail.tp_chars.tp_data
Implementation-dependent transaction program data that is passed unchanged on the DYNAMIC_LOAD_INDICATION.

tp_spec_data.pathname
Specifies the path and transaction program name.

tp_spec_data.parameters
Specifies the parameters for the transaction program.

tp_spec_data.queued
Specifies whether the transaction program will be queued.

tp_spec_data.load_type
Specifies how the transaction program will be loaded.

tp_spec_data.dynamic_load
Specifies whether the TP can be dynamically loaded (AP_YES or AP_NO).

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TP_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because the node has not yet been started, the Program returns the following parameters:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 7. Safe-Store Verbs

This chapter describes verbs that are issued at network nodes.

SAFE_STORE_TOPOLOGY

SAFE_STORE_TOPOLOGY is only used at a network node and safely stores topology information that can be later accessed if the node is restarted. The **restore** flag is used to indicate whether information is being stored (AP_NO) or accessed (AP_YES).

The store node information is returned as a formatted list. To obtain information about a specific network node or to obtain the list information in several chunks, the **index** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered on the **index_node_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM’s 6611 APPN MIB ordering). Next, the list is ordered on **index_node_type** by numeric value. If TGs are being stored or restored, ordering is on **index.tg_dest_node_name** (MIB ordering), then **index.tg_dest_node_type** (by numeric value), and thirdly on **index.tg_number** (by numeric value).

SAFE_STORE_TOPOLOGY verb supercedes the SFS_ADJACENT_NN, SFS_NN_TOPOLOGY_NODE and SFS_NN_TOPOLOGY_TG verbs. It stores topology information using control vectors as they appear in the topology, instead of translating to and from query overlays. Unknown control vectors are stored and restored, and a checksum is provided to prevent corrupt data from being introduced into the topology.

VCB Structure

```
typedef struct safe_store_topology
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;           /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
                                           /* to hold all information      */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  total_num_entries; /* total number of entries      */
        unsigned char   list_options;      /* listing options              */
        unsigned char   restore;           /* store or restore;            */
        unsigned char   resource_types;    /* resource types (nodes, TGs...)*/
        RESOURCE_INDEX  index;             /* resource index               */
        unsigned long   frsn;              /* flow-reduction sequence      */
                                           /* number                       */
        unsigned char   reserv3[16];       /* reserved                     */
      } SAFE_STORE_TOPOLOGY;

typedef struct resource_index
{
        unsigned char   node_name[17];     /* FQ node name                 */
        unsigned char   node_type;         /* node type                    */
        unsigned char   tg_dest_node_name[17];
                                           /* FQ name of TG destination node*/
        unsigned char   tg_dest_node_type;  /* TG destination node type      */
        unsigned char   tg_number;          /* TG number                     */
        unsigned char   reserv1[3];         /* reserved                      */
} RESOURCE_INDEX;
typedef struct safe_store_data
{
        unsigned short  overlay_size;       /* overalllength of safe         */
                                            /* store data                    */
        unsigned short  sub_overlay_size;   /* offset to first appended      */
                                            /* resource                      */
        RESOURCE_INDEX  index;              /* index of appended resource    */
        unsigned char   checksum[16];       /* reserved                      */
} SAFE_STORE_DATA;
typedef struct safe_store_node_data
{
        unsigned short  overlay_size;       /* overalllength of safe         */
                                            /* store data                    */
        unsigned short  sub_overlay_size;   /* offset to first appended      */
        unsigned char   adjacent;           /* is this NNCP and adjacent     */
                                            /* NNCP?                         */
        unsigned char   reserv1;            /* reserved                      */
        unsigned long   last_frsn_sent;     /* last flow reduction sequence  */
                                            /* num sent (if node is adjacent)*/
                                            /* resource                      */
       unsigned long   last_frsn_rcvd;      /* last flow reduction sequence  */
                                            /* num rcvd (if node is adjacent)*/
       unsigned long   frsn;                /* flow reduction sequence number*/
       unsigned short  days_left            /* days left in database         */
       unsigned short; vector_len           /* length of appended vector     */
} SAFE_STORE_NODE_DATA;
typedef struct safe_store_tg_data
{
        unsigned short  overlay_size;       /* overalllength of safe         */
                                            /* store data                    */
        unsigned short  sub_overlay_size;   /* offset to first appended      */
                                            /* resource                      */
        unsigned long   frsn;               /* flow reduction sequence number*/
        unsigned short  days_left           /* days left in database         */
        unsigned short  vector_len;         /* length of appended vector(s)  */
} SAFE_STORE_TG_DATA;
```

Supplied Parameters

Supplied Parameters when restore = AP_NO

The application supplies the following parameters:

opcode
AP_SAFE_STORE_TOPOLOGY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options

This indicates what should be returned in the list information. The **resource_types** and **index** specified (see the following parameters, **resource_types** and **index**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST

The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT

The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE

The returned list starts from the entry specified by the index value.

restore

Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

resource_types

This bit field controls the topology data to be stored. Any combination of the following values may be bit-wise ORed together in this field:

AP_SFS_NODES

Store topology nodes

AP_SFS_ADJ_NODES

Store adjacent nodes

AP_SFS_TGS

Store TGs

Note: At least one of these three flags must be set. Adjacent nodes and topology nodes are separate entities within APPN, so the first two flags can be set in any combination.

index.node_name

Network qualified node name from the Network Topology Database. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This field is also ignored if neither AP_SFS_NODES nor AP_SFS_ADJ_NODES is set in **resource_types**.

index.node_type

Type of the node. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN
AP_LEARN_NODE

If the **node_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This field is also ignored if neither AP_SFS_NODES nor AP_SFS_ADJ is set in **resource_types**.

index.tg_dest_node_name
Fully qualified destination node name for the TG. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This field is also ignored if neither AP_SFS_NODES nor AP_SFS_ADJ_NODES is set in **resource_types**.

index.tg_dest_node_type
Type of the the destiation node for this TG. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN

If the **tg_dest_node_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This field is also ignored if neither AP_SFS_TGS is not set in **resource_types**.

index.tg_number
The number associated with the TG. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST. This field is also ignored if neither AP_SFS_TGS is not set in **resource_types**.

frsn Flow Reduction Sequence Number (FRSN). If this is nonzero, then only topology resources with a FRSN greater than or equal to this value is returned.

safe_store_data.overlay_size
The length of this entry, including any padding. This is the offset to the next SAFE_STORE_DATA overlay, if any.

safe_store_data.sub_overlay_size
The length of this entry, including any padding. This is the offset to the appended SAFE_STORE_DATA or SAFE_STORE_TG_DATA. This field should always be used when accessing the appended data.

safe_store_data.index
The index for this entry. This structure can be supplied on subsequent SAFE_STORE_TOPOLOGY verbs to list subsequent entries. If **dest_tg_name** is set to all binary zeros, a SAFE_STORE_NODE_DATA overlay follows. Otherwise, a SAFE_STORE_TG_DATA overlay follows.

safe_store_data.checksum
The 128–bit checksum for the appended overlay and vectors. If this checksum and the following data becomes corrupted, it is highly probable that the corruption is detected and the verb is rejected.

safe_store_node_data.overlay_size
The length of this entry, including any padding. This is the offset to the appended SAFE_STORE_DATA or SAFE_STORE_TG_DATA.

safe_store_node_data.sub_overlay_size
The length of this entry, including any padding. This is the offset to the appended SAFE_STORE_DATA or SAFE_STORE_TG_DATA. This field should always be used to access the appended vectors.

safe_store_node_data.adjacent
AP_YES or AP_NO. If AP_YES, this entry corresponds to an adjacent Network Node.

safe_store_node_data.last_frsn_sent
If **adjacent** is set to AP_YES, this field holds the last FRSN sent to the adjacent Network Node. Otherwise, this field is set to zero.

safe_store_node_data.last_frsn_rcvd
If **adjacent** is set to AP_YES, this field holds the last FRSN sent to the adjacent Network Node. Otherwise, this field is set to zero.

safe_store_node_data.frsn
The Flow Reduction Sequence Number for this topology resource, if this node appears in the topology. Otherwise, this field is set to zero.

safe_store_node_data.days_left
The number of days this node remains in the topology database before being removed, unless its existence is can be confirmed. Zero signifies no limit.

safe_store_node_data.vector_len
The length of appended vectors. Zero signifies no vectors are appended.

safe_store_tg_data.overlay_size
The length of this entry, including any padding. This is the offset to the appended SAFE_STORE_DATA or SAFE_STORE_TG_DATA.

safe_store_tg_data.sub_overlay_size
The length of this entry, including any padding. This is the offset to the appended vectors, if there are any. This field should always be used to accessed appended vectors.

safe_store_tg_data.frsn
The Flow Reduction Sequence Number for this TG.

safe_store_tg_data.days_left
The number of days this TG remains in the topology database before being removed, unless its existence is can be confirmed. Zero signifies no limit.

safe_store_tg_data.vector_len
The length of appended vectors. Zero signifies no vectors are appended.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

num_entries
The number of entries actually returned.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LIST_OPTION

AP_INVALID_NODE
AP_INVALID_RESOURCE_TYPES
AP_INVALID_TG

Supplied Parameters

Supplied Parameters when restore = AP_YES

The application supplies the following parameters:

opcode
AP_SAFE_STORE_TOPOLOGY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

resource_types
This bit field controls the topology data to be stored. Any combination of the following values may be bit-wise ORed together in this field:

AP_SFS_NODES
Restore topology nodes

AP_SFS_ADJ_NODES
Restore adjacent nodes

AP_SFS_TGS
Restore TGs

Note: At least one of these three flags must be set. Adjacent nodes and topology nodes are separate entities within APPN, so the first two flags can be set in any combination.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_CHECKSUM_FAILED

AP_DATA_CORRUPT
AP_INVALID_RESOURCE_TYPES

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the system has not been built with Network Node support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the Node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SFS_ADJACENT_NN

Note: This verb has been superceded by SAFE_STORE_TOPOLOGY and is only retained for compatibility with previous versions of the Program.

SFS_ADJACENT_NN is used to safely store topology information that can be later accessed if the node is restarted. The **restore** flag is used to indicate whether information is being stored (AP_NO) or accessed (AP_YES).

When the **restore** flag is set to AP_NO, SFS_ADJACENT_NN returns information about adjacent network nodes (that is, those network nodes which CP-CP sessions are active, have been active, or have been active at some time).

The SFS information is returned as a formatted list. To obtain information about a specific network node or to obtain the list information in several chunks, the **adj_nncp_name** field should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See “Querying the Node” on page 10, for background on how the list formats are used.

This list is ordered on the **adj_nncp_name**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM’s 6611 APPN MIB ordering). If AP_LIST_FROM_NEXT is selected, the list starts from the next entry according to the defined ordering (whether the specified entry exists or not).

VCB Structure

```
typedef struct sfs_adjacent_nn
{
        unsigned short  opcode;             /* verb operation code          */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   format;             /* format                       */
        unsigned short  primary_rc;         /* primary return code          */
        unsigned long   secondary_rc;       /* secondary return code        */
        unsigned char   *buf_ptr;           /* pointer to buffer            */
        unsigned long   buf_size;           /* buffer size                  */
        unsigned long   total_buf_size;     /* total buffer size required   */
                                            /* to hold all information      */
        unsigned short  num_entries;        /* number of entries            */
        unsigned short  total_num_entries;  /* total number of entries      */
        unsigned char   list_options;       /* listing options              */
        unsigned char   restore;            /* store or restore;            */
        unsigned char   adj_nncp_name[17];  /* CP name of adj Network Node  */
} SFS_ADJACENT_NN;
typedef struct adj_nncp_data
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   adj_nncp_name[17];  /* CP name of adj Network Node  */
        unsigned char   cp_cp_sess_status;  /* CP-CP session status         */
        unsigned COUNTER
                        out_of_seq_tdus;    /* out of sequence TDUs         */
        unsigned long   last_frsn_sent;     /* last FSRN sent               */
        unsigned long   last_frsn_rcvd;     /* last FRSN received           */
        unsigned char   reserva[20];        /* reserved                     */
} ADJ_NNCP_DATA;
```

Supplied Parameters

Supplied Parameters when restore = AP_NO

The application supplies the following parameters:

opcode
AP_SFS_ADJACENT_NN

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **adj_nncp_name** specified (see the following parameter, **adj_nncp_name**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

adj_nncp_name
Fully-qualified, 17 byte, CP name of the adjacent network node composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

adj_nncp_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

adj_nncp_data.adj_nncp_name
This is a 17-byte fully-qualified CP name of adjacent network node which is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

adj_nncp_data.cp_cp_sess_status
Status of the CP-CP session (AP_ACTIVE or AP_INACTIVE).

adj_nncp_data.out_of_seq_tdus
Number of out of sequence TDUs received from this node.

adj_nncp_data.last_frsn_sent
The last flow reduction sequence number sent to this node.

adj_nncp_data.last_frsn_rcvd
The last flow reduction sequence number received from this node.

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

primary_rc
AP_OK

secondary_rc
AP_INVALID_ADJ_NNCP_NAME

AP_INVALID_LIST_OPTION

Supplied Parameters

Supplied Parameters when restore = AP_YES

The application supplies the following parameters:

opcode
AP_SFS_ADJACENT_NN

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

num_entries
The number of entries actually returned.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

adj_nncp_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

adj_nncp_data.adj_nncp_name
This is a 17-byte fully-qualified CP name of adjacent network node which is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

adj_nncp_data.cp_cp_sess_status
This field is ignored when **restore** is set to AP_YES.

adj_nncp_data.out_of_seq_tdus
This field is ignored when **restore** is set to AP_YES.

adj_nncp_data.last_frsn_sent
The last flow reduction sequence number sent to this node.

adj_nncp_data.last_frsn_rcvd
The last flow reduction sequence number received from this node.

Returned Parameters

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

primary_rc
AP_OK

If the verb does not execute because the node has not been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because one or more of the relevant START_NODE parameters were not sent, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the system is not built with network node support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SFS_DIRECTORY

In addition to the QUERY_DIRECTORY_ENTRY verb, there is the SFS_DIRECTORY verb that allows the local directory cache on a network node to be safely stored and can be later accessed if the node is restarted. The **restore** flag is used to indicate whether information is being stored (AP_NO) or accessed (AP_YES).

When the **restore** flag is set to AP_YES, SFS_DIRECTORY allows the directory database to be rebuilt using **directory_entry_summary** overlays. To obtain information about a specific network node or to obtain the list information in several chunks, the **resource_name** and **resource_type** fields should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

Resource information on the cached entries and their parents is returned in the following order:

```
1st Network Node
                           1st LU located at Network Node
                           2nd LU locate at Network Node
                           ...
                           nth LU located at Network Node
            1st End Node served by this Network Node
                           1st LU located at End Node(1)
                           2nd LU located at End Node(1)
                           ...
                           nth LU located at End Node(1)
             ...
             nth End Node served by this Network Node
                           1st LU located at End Node(n)
                           2nd LU located at End Node(n)
                           ...
2nd Network Node
            ...etc..
```

VCB Structure

```
typedef struct sfs_directory
{
        unsigned short  opcode;             /* verb operation code        */
        unsigned char   reserv2;            /* reserved                   */
        unsigned char   format;             /* format                     */
        unsigned short  primary_rc;         /* primary return code        */
        unsigned long   secondary_rc;       /* secondary return code      */
        unsigned char   *buf_ptr;           /* pointer to buffer          */
        unsigned long   buf_size;           /* buffer size                */
        unsigned long   total_buf_size;     /* total buffer size required */
        unsigned short  num_entries;        /* number of entries          */
        unsigned short  total_num_entries;  /* total number of entries    */
        unsigned char   list_options;       /* listing options            */
        unsigned char   restore;            /* store or restore flag      */
        unsigned char   resource_name[17];  /* network qualified res name */
        unsigned char   reserv3;            /* reserved                   */
        unsigned short  resource_type;      /* Resource type              */
       } SFS_DIRECTORY;
```

```
typedef struct directory_entry_summary
{
       unsigned short  overlay_size;        /* size of this entry          */
       unsigned char   resource_name[17];   /* network qualified res name  */
       unsigned char   reserve1;            /* reserved                    */
       unsigned short  resource_type;       /* Resource type               */
       unsigned short  real_owning_cp_type; /* real owning CP type         */
       unsigned char   real_owning_cp_name[17];
                                            /* real owning CP name         */
       unsigned char   description[RD_LEN]; /* resource description        */
} DIRECTORY_ENTRY_SUMMARY;
```

Supplied Parameters

Supplied Parameters when restore = AP_NO

The application supplies the following parameters:

opcode
AP_SFS_DIRECTORY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **resource_name** and **resource_type** specified (see the following parameters, **resource_name** and **resource_type**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composes of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

resource_type
Resource type. See one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

num_entries
The number of entries actually returned.

directory_entry_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

directory_entry_summary.resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composes of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

directory_entry_summary.resource_type
Resource type. See one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_RES_NAME

AP_INVALID_LIST_OPTION
AP_INVALID_RES_TYPE

directory_entry_summary.real_owning_cp_type
NN and BrNN only: Real owning CP type. This can be one of the following:

AP_NONE
The real owning CP is a parent resource.

AP_ENCP_RESOURCE
The real owning CP is not the parent resource and is an EN.

Other node types: This field is set to AP_NONE.

directory_entry_summary.real_owning_cp_name
NN and BrNN only: Fully qualified real owning CP name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If the real owning CP is the parent, this field is set to binary zeroes.

If the real owning CP is not the parent, then this field is set to the name of the real owning CP.

The real owning CP is not the parent in the directory of the NNS of a BrNN if the resource is owned by an EN in the domain of the BrNN. In this case, the real owning CP is the EN, but the parent is the BrNN.

Other node types: This field is set to binary zeroes.

Supplied Parameters

Supplied Parameters when restore = AP_YES

The application supplies the following parameters:

opcode
AP_SFS_DIRECTORY

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of the buffer supplied.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composes of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the application is restoring the first “chunk” of the directory, then this should be set to all zeros. Otherwise, the application should set this to the resource name of the last item in the previous “chunk”.

resource_type
Resource type. See one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

This field should be set to zero if the application is restoring the first "chunk" of the directory.

directory_entry_summary.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any). This must be the same as the **overlay_size** value returned when **restore** is set to AP_NO.

directory_entry_summary.resource_name
Network qualified resource name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composes of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

directory_entry_summary.resource_type
Resource type. See one of the following:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

directory_entry_summary.real_owning_cp_type
NN and BrNN only: Real owning CP type. This can be one of the following:

AP_NONE
The real owning CP is a parent resource.

AP_ENCP_RESOURCE
The real owning CP is not the parent resource and is an EN.

Other node types: This field is set to AP_NONE.

directory_entry_summary.real_owning_cp_name
NN and BrNN only: Fully qualified real owning CP name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

If the real owning CP is the parent, this field is set to binary zeroes.

If the real owning CP is not the parent, then this field is set to the name of the real owning CP.

The real owning CP is not the parent in the directory of the NNS of a BrNN if the resource is owned by an EN in the domain of the BrNN. In this case, the real owning CP is the EN, but the parent is the BrNN.

Other node types: This field is set to binary zeroes.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_RES_NAME

AP_INVALID_LIST_OPTION

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the system is not built with network node support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the Node has not been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SFS_NN_TOPOLOGY_NODE

Note: This verb has been superceded by SAFE_STORE_TOPOLOGY and is only retained for compatibility with previous versions of the Program.

Each network node maintains a network topology database that holds information about all network nodes, VRNs, and network node to network node TGs in the network. The SFS_NN_TOPOLOGY_NODE verb is used to safely store the topology database node entries that can be later accessed if the node is restarted. The **restore** flag is used to indicate whether information is being stored (AP_NO) or accessed (AP_YES).

To obtain information about a specific network node or to obtain the list information in several chunks, the **node_name** and **node_type** fields should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is by **node_name**, and **node_name_type**, and **frsn**. Ordering is by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). Ordering for the **node_type** is AP_NETWORK_NODE, then AP_VRN. The **frsn** is ordered numerically.

- If AP_LIST_INCLUSIVE is selected, the returned list starts from the first valid record of that name.
- If AP_LIST_FROM_NEXT is selected, the list will begin from the first valid record with a name following the one specified.

Note that if the **frsn** field is set to a nonzero value, only database entries with Flow Reduction Sequence Number (FRSNs) higher than this are returned. This allows a consistent topology database to be returned in a number of chunks by first getting the node's current FRSN. This works as follows:

1. Issue QUERY_NODE that returns the node's current FRSN.
2. Issue as many SFS_NN_TOPOLOGY_NODE (with FRSN set to zero) as necessary to get all the database entries in chunks.
3. Issue QUERY_NODE again and compare the new FRSN with the one returned in stage one.
4. If the two FRSNs are different then what has changed in the database, issue a SFS_NN_TOPOLOGY_NODE with the FRSN set to one greater than the FRSN supplied in stage one.

VCB Structure

```
typedef struct sfs_nn_topology_node
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  total_num_entries; /* total number of entries      */
        unsigned char   list_options;       /* listing options              */
        unsigned char   restore;            /* store or restore;            */
        unsigned char   node_name[17];      /* network qualified            */
                                            /* node name                    */
        unsigned char   node_type;          /* node type                    */
        unsigned long   frsn;               /* flow-reduction sequence      */
                                            /* number                       */
} SFS_NN_TOPOLOGY_NODE;

typedef struct nn_topology_node_detail
{
        unsigned short  overlay_size;       /* size of this entry           */
        unsigned char   node_name[17];      /* network qualified            */
        unsigned char   node_type;          /* node type                    */
        unsigned short  days_left           /* days left in database        */
        unsigned long   frsn;               /* flow reduction sequence number*/
        unsigned long   rsn;                /* resource sequence number     */
        unsigned char   rar;                /* route additional resistence  */
        unsigned char   status;             /* node status                  */
        unsigned char   function_support;   /* function support             */
        unsigned char   reserv2;            /* reserved                     */
        unsigned char   reserva[20];        /* reserved                     */
} NN_TOPOLOGY_NODE_DETAIL;
```

Supplied Parameters

Supplied Parameters when restore = AP_NO

The application supplies the following parameters:

opcode
AP_SFS_NN_TOPOLOGY_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **node_name, node_types** and **frsn** specified (see the following parameters, **node_name, node_types** and **frsn**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

node_name
Network qualified node name from the Network Topology Database. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

node_type
Type of the node. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN

If the **node_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

frsn Flow Reduction Sequence Number. If this is nonzero, then only topology resources with a FRSN greater than or equal to this value is returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

num_entries
The number of entries actually returned.

nn_topology_node_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nn_topology_node_detail.node_name
Network qualified node name from the Network Topology Database. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

nn_topology_node_detail.node_type
Type of the node. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_node_detail.days_left
Number of days before deletion of this node entry from the topology database. This will be set to zero for the local node entry (this entry is never deleted). This must be set to zero when the record is restored (for example, **restore** is set to AP_YES).

nn_topology_node_detail.frsn
The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

nn_topology_node_detail.rsn
The Resource Sequence Number. This is assigned by the network node that owns this resource.

nn_topology_node_detail.rar
The network node's route additional resistance.

nn_topology_node_detail.status
This field specifies the status of the node and can be AP_UNCONGESTED or one or more of the following ORed together:

AP_CONGESTED
The number of ISR sessions is greater than the **isr_sessions_upper_threshold** specified on the START_NODE verb.

AP_IRR_DEPLETED
The number of ISR sessions has reached the maximum specified on the **max_isr_sessions** parameter of the START_NODE verb.

AP_ERR_DEPLETED
The number of endpoint sessions has reached the maximum specified.

AP_QUIESCING
A STOP_NODE of type AP_QUIENCE or AP_QUIENCE_ISR was issued.

nn_topology_node_detail.function_support
This field specifies which functions are supported. This can be one or more of the following:

AP_BORDER_NODE
Border Node Function is supported.

AP_CDS
The Central Directory Server is supported.

AP_GATEWAY
The node is a Gateway Node (the function is not yet architecturally defined).

AP_ISR
This node supports the Intermediate Session Routing.

AP_HPR
This node supports the Intermediate Session Routing.

AP_RTP_TOWER
This node supports the RTP Tower of HPR.

AP_CONTROL_OVER_RTP_TOWER
This node supports the Control Flows Over the RTP Tower.

Note: The AP_CONTROL_OVER_RTP_TOWER node corresponds to the setting of both AP_HPR and AP_RTP_TOWER.

If the verb does not execute successfully, the Program returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LIST_OPTION

AP_INVALID_NODE
AP_INVALID_LIST_OPTIONS

Supplied Parameters

Supplied Parameters when restore = AP_YES

The application supplies the following parameters:

opcode
AP_SFS_NN_TOPOLOGY_NODE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

nn_topology_node_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nn_topology_node_detail.node_name
Network qualified node name from the Network Topology Database. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

nn_topology_node_detail.node_type
Type of the node. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_node_detail.days_left
Number of days before deletion of this node entry from the topology database. If the node is not the local node, this field must be set to a value greater than zero.

nn_topology_node_detail.frsn
The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

nn_topology_node_detail.rsn
The Resource Sequence Number. This is assigned by the network node that owns this resource.

nn_topology_node_detail.rar
The network node's route additional resistance.

nn_topology_node_detail.status
This field specifies the status of the node and can be AP_UNCONGESTED or one or more of the following ORed together:

AP_CONGESTED
The number of ISR sessions is greater than the **isr_sessions_upper_threshold** specified on the START_NODE verb.

AP_IRR_DEPLETED
The number of ISR sessions has reached the maximum specified on the **max_isr_sessions** parameter of the START_NODE verb.

AP_ERR_DEPLETED
The number of endpoint sessions has reached the maximum specified.

AP_QUIESCING
A STOP_NODE of type AP_QUIENCE or AP_QUIENCE_ISR was issued.

nn_topology_node_detail.function_support
This field specifies which functions are supported. This can be one or more of the following:

AP_BORDER_NODE
Border Node Function is supported.

AP_CDS
The Central Directory Server is supported.

AP_GATEWAY
The node is a Gateway Node (the function is not yet architecturally defined).

AP_ISR
This node supports the Intermediate Session Routing.

AP_HPR
This node supports the Intermediate Session Routing.

AP_RTP_TOWER
This node supports the RTP Tower of HPR.

AP_CONTROL_OVER_RTP_TOWER
This node supports the Control Flows Over the RTP Tower.

Note: The AP_CONTROL_OVER_RTP_TOWER node corresponds to the setting of both AP_HPR and AP_RTP_TOWER.

node_type
Type of the node. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN

If the **node_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

frsn Flow Reduction Sequence Number. If this is nonzero, then only topology resources with a FRSN greater than or equal to this value is returned.

Returned Parameters

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

primary_rc
AP_OK

secondary_rc
AP_INVALID_DAYS_LEFT

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DAYS_LEFT

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DAYS_LEFT

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameters:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

secondary_rc
AP_INVALID_DAYS_LEFT

If the verb does not execute because the system was not build with network node support, the Program returns the following parameters:

primary_rc
AP_INVALID_VERB

If the verb does not execute because of a system error, the Program returns the following parameters:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SFS_NN_TOPOLOGY_TG

Note: This verb has been superceded by SAFE_STORE_TOPOLOGY and is only retained for compatibility with previous versions of the Program.

Each network node maintains a network topology database that holds information about all network nodes, VRNs, and network node to network node TGs in the network. The SFS_NN_TOPOLOGY_TG verb is used to safely store the topology database node entries that can be later accessed if the node is restarted. The **restore** flag is used to indicate whether information is being stored (AP_NO) or accessed (AP_YES). The verb uses **topology_tg_detail** overlay.

To obtain information about a specific network node or to obtain the list information in several chunks, the **owner, owner_type, dest, dest_type,** and **tg_num** fields should be set.

Otherwise (if the **list_options** field is set to AP_FIRST_IN_LIST), this field will be ignored. See "Querying the Node" on page 10, for background on how the list formats are used.

This list is by **owner, owner_type, dest, dest_type, tg_num**, and **frsn**. The **owner_type** and **dest** name are ordered by name length first, and then by ASCII lexicographical ordering for names of the same length (in accordance with IBM's 6611 APPN MIB ordering). The ordering for **owner_type** and **dest** are: AP_ NETWORK_NODE, then AP_VRN. The **tg_num** and **frsn** is ordered numerically.

- If AP_LIST_INCLUSIVE is selected, the returned list starts from the first valid record of that name.
- If AP_LIST_FROM_NEXT is selected, the list will begin from the first valid record with a name following the one specified.

Note that if the **frsn** field is set to a nonzero value, only database entries with Flow Reduction Sequence Number (FRSNs) higher than this are returned. This allows a consistent topology database to be returned in a number of chunks by first getting the node's current FRSN. This works as follows:

1. Issue QUERY_NODE that returns the node's current FRSN.
2. Issue as many SFS_NN_TOPOLOGY_NODE (with FRSN set to zero) as necessary to get all the database entries in chunks.
3. Issue QUERY_NODE again and compare the new FRSN with the one returned in stage one.
4. If the two FRSNs are different then what has changed in the database, issue a SFS_NN_TOPOLOGY_NODE with the FRSN set to one greater than the FRSN supplied in stage one.

VCB Structure

```
typedef struct sfs_nn_topology_tg
{
        unsigned short  opcode;            /* verb operation code          */
        unsigned char   reserv2;           /* reserved                     */
        unsigned char   format;            /* format                       */
        unsigned short  primary_rc;        /* primary return code          */
        unsigned long   secondary_rc;      /* secondary return code        */
        unsigned char   *buf_ptr;          /* pointer to buffer            */
        unsigned long   buf_size;          /* buffer size                  */
        unsigned long   total_buf_size;    /* total buffer size required   */
        unsigned short  num_entries;       /* number of entries            */
        unsigned short  total_num_entries;  /* total number of entries       */
        unsigned char   list_options;       /* listing options               */
        unsigned char   restore;            /* store or restore;             */
        unsigned char   owner[17];          /* network qualified             */
                                            /* node name                     */
        unsigned char   owner_type;         /* node type                     */
        unsigned char   dest[17];           /* TG destination node           */
        unsigned char   dest_type;          /* TG destination node type      */
        unsigned char   tg_num;             /* TG number                     */
        unsigned char   reserv1;            /* reserved                      */
        unsigned long   frsn;               /* flow-reduction sequence       */
                                            /* number                        */
} SFS_NN_TOPOLOGY_TG;

typedef struct nn_topology_tg_detail
{
        unsigned short  overlay_size;       /* size of this entry            */
        unsigned char   owner[17];          /* network qualified             */
        unsigned char   owner_type;         /* node type                     */
        unsigned char   dest[17];           /* TG destination node           */
        unsigned char   dest_type           /* TG destination node type      */
        unsigned char   tg_num;             /* TG number                     */
        unsigned char   reserv3[1];         /* reserved                      */
        unsigned long   frsn;               /* flow reduction sequence number*/
        unsigned short  days_left           /* days left in database         */
        LINK_ADDRESS    dlc_data;           /* DLC signalling data           */
        unsigned long   rsn;                /* resource sequence number      */
        unsigned char   status;             /* node status                   */
        TG_DEFINED_CHAR tg_chars;           /* TG characteristics            */
        unsigned char   reserva[20];        /* reserved                     */
}TOPOLOGY_TG_DETAIL;

typedef struct link_address
{
        unsigned short  length              /* length                        */
        unsigned short  reserve1;           /* reserved                      */
        unsigned char   address[MAX_LINK_ADDR_LEN];
                                            /* address                       */
}LINK_ADDRESS;
```

Supplied Parameters

Supplied Parameters when restore = AP_NO

The application supplies the following parameters:

opcode
AP_SFS_NN_TOPOLOGY_TG

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **owner,**

owner_type, dest, dest_type, tg_num, and **frsn** specified (see the following parameters, **owner, owner_type, dest, dest_type, tg_num**, and **frsn**) represents an index value that is used to specify the starting point of the actual information to be returned.

AP_FIRST_IN_LIST
: The index value is ignored, and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
: The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
: The returned list starts from the entry specified by the index value.

restore
: Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

owner
: Name of the TG's originating node (always set to the local node name). This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

owner_type
: Type of the node. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN

If the **owner_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dest
: Fully qualified destination node name for the TG. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

dest_type
: Type of the node. This node is set to one of the following:

AP_NETWORK_NODE
AP_VRN

If the **dest_type** is unknown, AP_LEARN_NODE must be specified. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

tg_num
: Number associated with the TG. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

frsn
: Flow Reduction Sequence Number. If this is nonzero, then only topology resources with a FRSN greater than or equal to this value is returned.

Returned Parameters

If the verb executes successfully, the Program 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 can be higher than **buf_size**.

num_entries
The number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

nn_topology_tg_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

nn_topology_detail.owner
Name of the TG's originating node. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

nn_topology_tg_detail.owner_type
Type of the node. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_tg_detail.dest
Fully qualified destination node name for the TG. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

nn_topology_tg_detail.dest_type
Type of the node. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_tg_detail.tg_num
The number associated with the TG.

nn_topology_tg_detail.frsn
The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

nn_topology_tg_detail.days_left
The number of days this node remains in the topology database before

being removed, unless its existence is can be confirmed. If the node specified by the **owner** field is not the local node, this field must be set to a value greater than zero.

nn_topology_tg_detail.dlc_data.length
The address length.

nn_topology_tg_detail.dlc_data.address
The address.

nn_topology_tg_detail.rsn
The Resource Sequence Number. This is assigned by the network node that owns this resource.

nn_topology_tg_detail.status
This field specifies the status of the TG. This can be one or more of the following ORed together:

AP_TG_OPERATIVE
AP_TG_CP_CP_SESSIONS
AP_TG_QUIESCING
AP_TG_HPR
AP_TG_RTP
AP_NONE

nn_topology_tg_detail.tg_chars
The TG characteristics. See "DEFINE_CN" on page 31 for additional information.

Returned Parameters

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_TG

AP_INVALID_ORIGIN_NODE
AP_INVALID_LIST_OPTION

If the verb does not execute successfully, the Program returns the following parameters:

primary_rc
AP_OK

Supplied Parameters

Supplied Parameters when restore = AP_YES

This application supplies the following parameters:

opcode
AP_SFS_NN_TOPOLOGY_TG

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer where list information can be written. The application can append data to the end of the VCB, in which case **buf_ptr** must be set to NULL.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

buf_size
Length of the information returned in the buffer.

restore
Flag indicating whether the information should be restored (AP_YES) or stored (AP_NO). In this case, it is set to AP_NO.

total_num_entries
Total number of entries that could have been returned. This can be higher than **num_entries**.

nn_topology_tg_detail.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any). This must be the same as the **overlay_size** value returned when **restore** = AP_NO.

nn_topology_detail.owner
Name of the TG's originating node. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

nn_topology_tg_detail.owner_type
Type of the node that owns the TG. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_tg_detail.dest
Fully qualified destination node name for the TG. This name is a 17-byte adjacent control point name, which is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only relevant for links to APPN nodes and is otherwise ignored. This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

nn_topology_tg_detail.dest_type
Type of the node. It is one of the following:

AP_NETWORK_NODE
AP_VRN

nn_topology_tg_detail.tg_num
The number associated with the TG.

nn_topology_tg_detail.frsn
The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

nn_topology_tg_detail.days_left
The number of days this node remains in the topology database before

being removed, unless its existence is can be confirmed. If the node specified by the **owner** field is not the local node, this field must be set to a value greater than zero.

nn_topology_tg_detail.dlc_data.length
The address length.

nn_topology_tg_detail.dlc_data.address
The address.

nn_topology_tg_detail.rsn
The Resource Sequence Number. This is assigned by the network node that owns this resource.

nn_topology_tg_detail.status
This field specifies the status of the TG. This can be one or more of the following ORed together:

AP_TG_OPERATIVE
AP_TG_CP_CP_SESSIONS
AP_TG_QUIESCING
AP_TG_HPR
AP_TG_RTP
AP_NONE

nn_topology_tg_detail.tg_chars
The TG characteristics. See "DEFINE_CN" on page 31 for additional information.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_DAYS_LEFT

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the system was not built with the network node support, the Program returns the following parameter:

primary_rc
AP_INVALID_VERB

If the verb does not execute because the node has not been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSYEM_ERROR

Chapter 8. Session Limit Verbs

This chapter describes verbs used to initialize, change, or reset session limits.

CHANGE_SESSION_LIMIT

The CHANGE_SESSION_LIMIT verb requests that the session limits of a particular mode (or session group) be changed. Sessions can be activated or deactivated as a result of processing this verb.

VCB Structure

```
typedef struct change_session_limit
{
   unsigned short  opcode;             /* verb operation code      */
   unsigned char   reserv2;            /* reserved                 */
   unsigned char   format;             /* format                   */
   unsigned short  primary_rc;         /* primary return code      */
   unsigned long   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? */
   unsigned short  plu_mode_session_limit;
                                       /* session limit            */
   unsigned short  min_conwinners_source;
                                       /* min source contention    */
                                       /* winner sessions          */
   unsigned short  min_conwinners_target;
                                       /* min target contention    */
                                       /* winner sessions          */
   unsigned short  auto_act;           /* auto activation limit    */
   unsigned char   responsible;        /* responsible indicator    */
   unsigned char   reserv4[3];         /* reserved                 */
   unsigned long   sense_data;         /* sense data               */
} CHANGE_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_CHANGE_SESSION_LIMIT

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to change session limits. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to change session limits. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** fields are set to all zeros then the verb is forwarded to the LU associated with the control point (the default LU).

plu_alias
: Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

fqplu_name
: Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

mode_name
: Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

  SNASVCMG and CPSVCMG mode limits cannot be changed. **Set_negotiable** specifies whether the maximum negotiable session limit for this mode should be modified to become the **plu_mode_session_limit**.

set_negotiable
: Specifies whether the maximum negotiable session limit for this mode should be modified to become the **plu_mode_session_limit**.

  AP_YES
  AP_NO

plu_mode_session_limit
: Requested total session limit for this mode. The actual session limit (which can be negotiated with the partner LU), is the agreed maximum number of sessions supported between the local LU and the partner LU on this mode.

min_conwinners_source
: Minimum number of sessions in this mode for which the local LU is the contention winner.

min_conwinners_target
: Minimum number of sessions in this mode for which the partner LU is the contention winner.

auto_act
: Number of sessions to automatically activate after the session limit is changed. 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 source (local) or target (partner) LU is responsible for deactivating sessions after the session limit is changed (AP_SOURCE or AP_TARGET).

Returned Parameters

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

primary_rc
AP_OK

secondary_rc
AP_AS_SPECIFIED

AP_AS_NEGOTIATED

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_LU_MODE_SESSION_LIMIT_ZERO

AP_EXCEEDS_MAX_ALLOWED
AP_INVALID_MODE_NAME
AP_INVALID_PLU_NAME
AP_INVALID_RESPONSIBLE
AP_INVALID_SET_NEGOTIABLE
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_MODE_RESET

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of an allocation error, the Program returns the following parameters:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY

sense_data
Sense data associated with allocation error.

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

If the verb does not execute because of an error, the Program returns the following parameters:

primary_rc

AP_CONV_FAILURE_NO_RETRY
AP_CNOS_PARTNER_LU_REJECT

secondary_rc
AP_CNOS_COMMAND_RACE_REJECT

AP_CNOS_MODE_NAME_REJECT

INITIALIZE_SESSION_LIMIT

The INITIALIZE_SESSION_LIMIT verb initializes the mode session limits.

VCB Structure

```
typedef struct initialize_session_limit
{
   unsigned short  opcode;              /* verb operation code       */
   unsigned char   reserv2;             /* reserved                  */
   unsigned char   format;              /* format                    */
   unsigned short  primary_rc;          /* primary return code       */
   unsigned long   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? */
   unsigned short  plu_mode_session_limit;
                                        /* session limit             */
   unsigned short  min_conwinners_source;
                                        /* min source contention     */
                                        /* winner sessions           */
   unsigned short  min_conwinners_target;
                                        /* min target contention     */
                                        /* winner sessions           */
   unsigned short  auto_act;            /* auto activation limit     */
   unsigned char   reserv4[4];          /* reserved                  */
   unsigned long   sense_data;          /* sense data                */
} INITIALIZE_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_INITIALIZE_SESSION_LIMIT

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to initialize session limits. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to initialize session limits. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and **lu_alias** are set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

plu_alias
Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are

significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

fqplu_name
Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

mode_name
Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

This verb is rejected if one of the mode names SNASVCMG or CPSVCMG is supplied in this field and limits take values other than **plu_mode_session_limit 2, min_conwinners_source 1,** and **min_conwinners target 1**.

set_negotiable
Specifies whether the maximum negotiable session limit for this mode should be modified to become the **plu_mode_session_limit**.

AP_YES
AP_NO

plu_mode_session_limit
Requested total session limit for this mode. The actual session limit (which can be negotiated with the partner LU), is the agreed maximum number of sessions supported between the local LU and the partner LU on this mode. This must be set to a value in the range one to 32 767.

min_conwinners_source
Minimum number of sessions in this mode for which the local LU is the contention winner. This must be set to a value in the range zero to 32 767.

min_conwinners_target
Minimum number of sessions in this mode for which the partner LU is the contention winner. This must be set to a value in the range zero to 32 767.

auto_act
Number of sessions to automatically activate after the session limit is changed. 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. This must be set to a value in the range zero to 32 767.

Returned Parameters

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

primary_rc
AP_OK

secondary_rc
AP_AS_SPECIFIED

AP_AS_NEGOTIATED

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_CANT_CHANGE_TO_ZERO

AP_EXCEEDS_MAX_ALLOWED
AP_INVALID_SET_NEGOTIABLE
AP_INVALID_PLU_NAME
AP_INVALID_MODE_NAME
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS
AP_INVALID_SCVMG_LIMITS

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_MODE_NOT_RESET

If the verb does not execute because the node has not yet been started, the Program returns the following parameters:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of an allocation error, the Program returns the following parameters:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY

sense_data
Sense data associated with allocation error.

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

If the verb does not execute because of an error, the Program returns the following parameters:

primary_rc

AP_CONV_FAILURE_NO_RETRY
AP_CNOS_PARTNER_LU_REJECT

secondary_rc
AP_CNOS_COMMAND_RACE_REJECT

AP_CNOS_MODE_NAME_REJECT

RESET_SESSION_LIMIT

The RESET_SESSION_LIMIT verb requests that the mode session limits be reset.

VCB Structure

```
typedef struct reset_session_limit
{
   unsigned short  opcode;           /* verb operation code       */
   unsigned char   reserv2;          /* reserved                  */
   unsigned char   format;           /* format                    */
   unsigned short  primary_rc;       /* primary return code       */
   unsigned long   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 qual 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?  */
   unsigned char   reserv4[8];       /* reserved                  */
   unsigned char   responsible;      /* responsible               */
   unsigned char   drain_source;     /* drain source              */
   unsigned char   drain_target;     /* drain target              */
   unsigned char   force;            /* force                     */
   unsigned long   sense_data;       /* sense data                */
} RESET_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_RESET_SESSION_LIMIT

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU requested to reset session limits. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias of the local LU requested to reset session limits. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If this is set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

plu_alias
Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

fqplu_name
Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can

have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

mode_name
Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

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.

AP_ONE
AP_ALL

set_negotiable
Specifies whether the maximum negotiable session limit for this mode should be modified.

AP_YES
AP_NO

responsible
Indicates whether the source (local) or target (partner) LU is responsible for deactivating sessions after the session limit is reset (AP_SOURCE or AP_TARGET).

drain_source
Specifies whether the source LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

drain_target
Specifies whether the target LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

force Specifies whether session limits will be set to zero even if CNOS negotiation fails (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

secondary_rc
AP_FORCED

AP_AS_SPECIFIED
AP_AS_NEGOTIATED

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_EXCEEDS_MAX_ALLOWED

AP_INVALID_PLU_NAME
AP_INVALID_MODE_NAME
AP_INVALID_MODE_NAME_SELECT
AP_INVALID_RESPONSIBLE
AP_INVALID_DRAIN_SOURCE
AP_INVALID_DRAIN_TARGET
AP_INVALID_FORCE
AP_INVALID_SET_NEGOTIABLE
AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_MODE_RESET

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of an allocation error, the Program returns the following parameter:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY

sense_data
Sense data associated with allocation error.

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

If the verb does not execute because of an error, the Program returns the following parameters:

primary_rc
AP_CONV_FAILURE_NO_RETRY

AP_CNOS_PARTNER_LU_REJECT

secondary_rc
AP_CNOS_COMMAND_RACE_REJECT

AP_CNOS_MODE_NAME_REJECT

Chapter 9. Node Operator Facility API Indications

The Node Operator Facility API generates indication verbs to notify a node operator about changes in the node. Indication verbs use the following general structure:

```
typedef struct indication_hdr
           {
             unsigned short  opcode;         /* verb operation code      */
             unsigned char   reserv2;        /* reserved                 */
             unsigned char   format;         /* format                   */
             unsigned short  primary_rc;     /* primary return code      */
             unsigned long   secondary_rc;   /* secondary return code    */
             unsigned char   data_lost;      /* previous indication lost */
           } INDICATION_HDR;
```

DLC_INDICATION

This indication is generated when the DLC goes from active to inactive, or from inactive to active.

VCB Structure

```
typedef struct dlc_indication
{
   unsigned short  opcode;              /* verb operation code           */
   unsigned char   attributes;          /* verb attributes               */
   unsigned char   format;              /* format                        */
   unsigned short  primary_rc;          /* primary return code           */
   unsigned long   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[RD_LEN]; /* resource description          */
   unsigned char   reserva[20];         /* reserved                      */
} DLC_INDICATION;
```

Parameters

opcode
AP_DLC_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_YES when the DLC becomes inactive. Set to AP_NO when the DLC becomes active.

dlc_name
Name of DLC. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

description
Resource description (as specified on DEFINE_DLC). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

DLUR_LU_INDICATION

This indication is generated whenever a DLUR LU is activated or deactivated. This allows a registered application to maintain a list of currently active DLUR LUs.

VCB Structure

```
typedef struct dlur_lu_indication
{
   unsigned short  opcode;              /* verb operation code            */
   unsigned char   reserv2;             /* reserved                       */
   unsigned char   format;              /* format                         */
   unsigned short  primary_rc;          /* primary return code            */
   unsigned long   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[8];          /* PU name                        */
   unsigned char   nau_address;         /* NAU address                    */
   unsigned char   reserv5[7];          /* reserved                       */
} DLUR_LU_INDICATION;
```

Parameters

opcode
AP_DLUR_LU_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

reason Set to AP_ADDED if the DLUR LU has just been activated by the DLUS. Set to AP_REMOVED if the DLUR LU 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 LU. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

pu_name
Name of the PU that this LU uses. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

nau_address
Network addressable unit address of the LU, which must be in the range 1–255.

DLUR_PU_INDICATION

This indication is generated whenever a DLUR PU is activated or deactivated. This allows a registered application to maintain a list of currently active DLUR PUs.

VCB Structure

```
typedef struct dlur_pu_indication
{
   unsigned short  opcode;             /* verb operation code            */
   unsigned char   reserv2;            /* reserved                       */
   unsigned char   format;             /* format                         */
   unsigned short  primary_rc;         /* primary return code            */
   unsigned long   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

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

reason The cause of the indication. It is one of the following:

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 PU. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

pu_id
The PU identifier defined in a DEFINE_INTERNAL_PU verb or obtained in an XID from a downstream PU. This 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
The location of the PU. This can be one of the following:

AP_INTERNAL
AP_DOWNSTREAM

pu_status
The status of the PU (as seen by DLUR). This can be set to one of the following:

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 be retried.

AP_PEND_ACTPU
The PU is waiting for an ACTPU from the host.

AP_PEND_ACTPU_RSP
After forwarding an ACTPU to the PU, DLUR is waiting for the PU to respond.

AP_ACTIVE
The PU is activate.

AP_PEND_DACTPU_RSP
After forwarding an DACTPU to the PU, DLUR is waiting for the PU to respond.

AP_PEND_INOP
DLUR is waiting for all necessary events to complete before it deactivates the PU.

dlus_name
The name of the DLUS node that the PU is currently using (or attempting to use). This is a 17–byte string composed of two type A EBCDIC character strings concatenated by an EBCDIC dot, that is right padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.) If the PU activation has failed, this field will be set to all zeros.

dlus_session_status
The status of the DLUS pipe currently being used by the PU. This can be one of the following:

AP_PENDING_ACTIVE
AP_ACTIVE
AP_PENDING_INACTIVE
AP_INACTIVE

DLUS_INDICATION

This indication is generated when a pipe to a DLUS node goes from inactive to active (or vice versa). Pipe statistics are supplied when the pipe becomes inactive.

VCB Structure

```
typedef struct dlus_indication
{
   unsigned short  opcode;             /* verb operation code            */
   unsigned char   reserv2;            /* reserved                       */
   unsigned char   format;             /* format                         */
   unsigned short  primary_rc;         /* primary return code            */
   unsigned long   secondary_rc;       /* secondary return code          */
   unsigned char   data_lost;          /* previous indication lost       */
   unsigned char   deactivated;        /* has session been deactivated?  */
   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
{
   unsigned long   reqactpu_sent;      /* REQACTPUs sent to DLUS         */
   unsigned long   reqactpu_rsp_received;
                                       /* RSP(REQACTPU)s received        */
                                       /* from DLUS                      */
   unsigned long   actpu_received;     /* ACTPUs received from DLUS      */
   unsigned long   actpu_rsp_sent;     /* RSP(ACTPU)s sent to DLUS       */
   unsigned long   reqdactpu_sent;     /* REQDACTPUs sent to DLUS        */
   unsigned long   reqdactpu_rsp_received;
                                       /* RSP(REQDACTPU)s received       */
                                       /* from DLUS                      */
   unsigned long   dactpu_received;    /* DACTPUs received from DLUS     */
   unsigned long   dactpu_rsp_sent;    /* RSP(DACTPU)s sent to DLUS      */
   unsigned long   actlu_received;     /* ACTLUs received from DLUS      */
   unsigned long   actlu_rsp_sent;     /* RSP(ACTLU)s sent to DLUS       */
   unsigned long   dactlu_received;    /* DACTLUs received from DLUS     */
   unsigned long   dactlu_rsp_sent;    /* RSP(DACTLU)s sent to DLUS      */
   unsigned long   sscp_pu_mus_rcvd;   /* MUs for SSCP-PU sess received  */
   unsigned long   sscp_pu_mus_sent;   /* MUs for SSCP-PU sessions sent  */
   unsigned long   sscp_lu_mus_rcvd;   /* MUs for SSCP-LU sess received  */
   unsigned long   sscp_lu_mus_sent;   /* MUs for SSCP-LU sessions sent  */
} PIPE_STATS;
```

Parameters

opcode
AP_DLUS_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication

to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_YES when the pipe becomes inactive. Set to AP_NO when the pipe becomes active.

dlus_name
Name of the DLUS. This is a 17-byte string composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, which is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

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 verb applies only to Communications Server.

This indication is generated when the LU-SSCP session between the downstream LU and the host goes from inactive to active (or vice-versa) or when the PLU-SLU session goes from inactive to active (or vice-versa). LU-SSCP statistics are supplied when the LU-SSCP session deactivates and PLU-SLU statistics are supplied when the PLU-SLU session deactivates.

VCB Structure

```
typedef struct downstream_lu_indication
{
   unsigned short  opcode;                /* verb operation code           */
   unsigned char   attributes;            /* attributes                    */
   unsigned char   format;                /* format                        */
   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[RD_LEN];   /* resource description          */
   unsigned char   nau_address;           /* NAU address                   */
   unsigned char   lu_sscp_sess_active;
                                          /* Is 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 sess stats */
   SESSION_STATS   us_plu_stats;          /* Upstream PLU-SLU sess 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;     /* max send BTU size             */
   unsigned short  max_rcv_btu_size;      /* max rcv BTU size              */
   unsigned short  max_send_pac_win;      /* max send pacing window size   */
   unsigned short  cur_send_pac_win;      /* curr send pacing window size  */
   unsigned short  max_rcv_pac_win;       /* max rcv pacing window size    */
   unsigned short  cur_rcv_pac_win;       /* curr receive pacing win 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;       /* num of data frames received   */
   unsigned long   rcv_fmd_data_frames;
                                          /* num FMD data frames received  */
   unsigned long   rcv_data_bytes;        /* num 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[8];            /* Link station name             */
   unsigned char   pacing_type;           /* type of pacing in use         */
} SESSION_STATS;
```

Parameters

opcode
AP_DOWNSTREAM_LU_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been 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 to the right with EBCDIC spaces.

ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

dslu_name
Name of the downstream LU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
Resource description (as specified on DEFINE_DOWNSTREAM_LU).

nau_address
Network addressable unit address of the LU which must be in the range 1–255.

lu_sscp_sess_active
Indicates whether the LU-SSCP session to the downstream LU is active. Set to either AP_YES or AP_NO.

plu_sess_active
Indicates whether the PLU-SLU session to the downstream LU is active. Set to either AP_YES or AP_NO.

dspu_services
Specifies the services which the local node provides to the downstream LU across the link. This is set to one of the following.

AP_PU_CONCENTRATION
Local node provides PU concentration for the downstream PU.

AP_DLUR
Local node provides DLUR support for the downstream PU.

lu_sscp_stats.rcv_ru_size
This field is always reserved.

lu_sscp_stats.send_ru_size
This field is always reserved.

lu_sscp_stats.max_send_btu_size
Maximum BTU size that can be sent.

lu_sscp_stats.max_rcv_btu_size
Maximum BTU size that can be received.

lu_sscp_stats.max_send_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_rcv_pac_win
This field will always be 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 FMD data 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 FMD data 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 address 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 1 if the BIND 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 8 bytes are significant.

lu_sscp_stats.pacing_type
: Receive pacing type in use on the upstream LU-SSCP session. This will take the value AP_NONE.

ds_plu_stats.rcv_ru_size
: Maximum receive RU size.

ds_plu_stats.send_ru_size
: Maximum send RU size.

ds_plu_stats.max_send_btu_size
: Maximum BTU size that can be sent.

ds_plu_stats.max_rcv_btu_size
: Maximum BTU size that can be received.

ds_plu_stats.max_send_pac_win
: Maximum size of the send pacing window on this session.

ds_plu_stats.cur_send_pac_win
: Current size of the send pacing window on this session

ds_plu_stats.max_rcv_pac_win
: Maximum size of the receive pacing window on this session.

ds_plu_stats.cur_rcv_pac_win
: Current size of the receive pacing window on this session.

ds_plu_stats.send_data_frames
: Number of normal flow data frames sent.

ds_plu_stats.send_fmd_data_frames
: Number of normal flow FMD data frames sent.

ds_plu_stats.send_data_bytes
: Number of normal flow data bytes sent.

ds_plu_stats.rcv_data_frames
: Number of normal flow data frames received.

ds_plu_stats.rcv_fmd_data_frames
: Number of normal flow FMD data frames received.

ds_plu_stats.rcv_data_bytes
: Number of normal flow data bytes received.

ds_plu_stats.sidh
: Session ID high byte.

ds_plu_stats.sidl
: Session ID low byte.

ds_plu_stats.odai
: Origin destination address 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 1 if the BIND sender is the node containing the secondary link station.

ds_plu_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.

ds_plu_sscp_stats.pacing_type
: Receive pacing type in use on the downstream PLU-SLU session. This can be set to AP_NONE or AP_PACING_FIXED.

us_plu_stats.rcv_ru_size
Maximum receive RU size.

us_plu_stats.send_ru_size
Maximum send RU size.

us_plu_stats.max_send_btu_size
Maximum BTU size that can be sent.

us_plu_stats.max_rcv_btu_size
Maximum BTU size that can be received.

us_plu_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

us_plu_stats.cur_send_pac_win
Current size of the send pacing window on this session

us_plu_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

us_plu_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

us_plu_stats.send_data_frames
Number of normal flow data frames sent.

us_plu_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

us_plu_stats.send_data_bytes
Number of normal flow data bytes sent.

us_plu_stats.rcv_data_frames
Number of normal flow data frames received.

us_plu_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

us_plu_stats.rcv_data_bytes
Number of normal flow data bytes received.

us_plu_stats.sidh
Session ID high byte. This field is reserved if **dspu_services** is set to AP_PU_CONCENTRATION.

us_plu_stats.sidl
Session ID low byte. This field is reserved if **dspu_services** is set to AP_PU_CONCENTRATION.

us_plu_stats.odai
Origin destination address 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 1 if the BIND sender is the node containing the secondary link station. This field is reserved if **dspu_services** is set to AP_PU_CONCENTRATION.

us_plu_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 is reserved if **dspu_services** is set to AP_PU_CONCENTRATION.

us_plu_stats.pacing_type
Receive pacing type in use on the upstream PLU-SLU session. This can take the values AP_NONE or AP_PACING_FIXED.

DOWNSTREAM_PU_INDICATION

This verb applies only to Communications Server.

This indication is generated when the PU-SSCP session between the downstream PU and the host goes from inactive to active (or vice-versa). PU-SSCP statistics are supplied when the PU-SSCP session deactivates.

VCB Structure

```
typedef struct downstream_pu_indication
{
   unsigned short  opcode;               /* verb operation code          */
   unsigned char   attributes;           /* attributes                   */
   unsigned char   format;               /* format                       */
   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[RD_LEN];  /* resource description         */
   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;    /* max send BTU size            */
   unsigned short  max_rcv_btu_size;     /* max rcv BTU size             */
   unsigned short  max_send_pac_win;     /* max send pacing window size  */
   unsigned short  cur_send_pac_win;     /* curr send pacing window size */
   unsigned short  max_rcv_pac_win;      /* max rcv pacing window size   */
   unsigned short  cur_rcv_pac_win;      /* curr receive pacing win 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;      /* num of data frames received  */
   unsigned long   rcv_fmd_data_frames;
                                         /* num FMD data frames received */
   unsigned long   rcv_data_bytes;       /* num 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[8];           /* Link station name            */
   unsigned char   pacing;               /* pacing_type                    */
} SESSION_STATS;
```

Parameters

opcode
AP_DOWNSTREAM_PU_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

dspu_name
Name of the downstream PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
Resource description (as specified on DEFINE_LS.

ls_name
Name of link station. This is a 8-byte string in a locally displayable character set. All 8 bytes are significant.

pu_sscp_sess_active
Indicates whether the PU-SSCP session to the downstream PU is active. Set to either AP_YES or AP_NO.

dspu_services
Specifies the services which the local node provides to the downstream PU across the link. This is set to one of the following.

AP_PU_CONCENTRATION
Local node provides PU concentration for the downstream PU.

AP_DLUR
Local node provides DLUR support for the downstream PU.

pu_sscp_stats.rcv_ru_size
This field is always reserved.

pu_sscp_stats.send_ru_size
This field is always reserved.

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
This field will always be set to zero.

pu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

pu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

pu_sscp_stats.cur_rcv_pac_win
This field will always be 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 address 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 1 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 string in a locally displayable character set. All 8 bytes are significant.

pu_sscp_stats.pacing_type
Receive pacing type in use on the upstream PU-SSCP session. This will take the value AP_NONE.

FOCAL_POINT_INDICATION

This indication is generated whenever a focal point is acquired, changed or revoked.

VCB Structure

```
typedef struct focal_point_indication
{
   unsigned short  opcode;            /* verb operation code          */
   unsigned char   reserv2;           /* reserved                     */
   unsigned char   format;            /* format                       */
   unsigned short  primary_rc;        /* primary return code          */
   unsigned long   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

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

ms_category
Category of focal point where the focal point has been acquired, changed or revoked. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services categories as described in *SNA Management Services* , or an 8-byte type 1134 EBCDIC installation defined name.

fp_fqcp_name
The fully qualified control point name of the current focal point. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This name will be all zeros if the focal point has been revoked and not replaced (so that there is no currently active focal point).

ms_appl_name
Name of the current focal point application. This can either be one of the 4-byte architecturally defined values (right-padded with EBCDIC spaces) for management services applications as described in *SNA Management Services* , or an 8-byte type-1134 EBCDIC installation defined name. This will be all zeros if the focal point has been revoked and not replaced (so that there is no currently active focal point).

fp_type
Type of focal point. Refer to *SNA Management Services* for further details.

AP_EXPLICIT_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:

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 using MDS transport to send data to the focal point (only significant if the focal point status is AP_ACTIVE):

AP_DEFAULT
Default routing is used to deliver the MDS_MU to the focal point.

AP_DIRECT
The MDS_MU will be routed on a session directly to the focal point.

ISR_INDICATION

This verb applies only to Communications Server.

This indication is generated when an ISR session is activated or deactivated. When the session is deactivated, final session statistics are returned. When the session is activated the **pri_sess_stats** and **sec_sess_stats** fields are reserved.

VCB Structure

```
typedef struct isr_indication
{
   unsigned short  opcode;             /* verb operation code          */
   unsigned char   reserv2;            /* reserved                     */
   unsigned char   format;             /* format                       */
   unsigned short  primary_rc;         /* primary return code          */
   unsigned long   secondary_rc;       /* secondary return code        */
   unsigned char   data_lost;          /* previous indication lost     */
   unsigned char   deactivated;        /* has ISR session been         */
                                       /* deactivated?                 */
   FQPCID          fqpcid;             /* fully qualified procedure    */
                                       /* correlator ID                */
   unsigned char   fqplu_name[17];     /* fully qualified primary      */
                                       /* LU name                      */
   unsigned char   fqslu_name[17];     /* fully qualified secondary    */
                                       /* LU name                      */
   unsigned char   mode_name[8];       /* mode name                    */
   unsigned char   cos_name[8];        /* COS name                     */
   unsigned char   transmission_priority;
                                       /* transmission priority        */
   unsigned long   sense_data;         /* sense data                   */
   unsigned char   reserv2a[2];        /* reserved                     */
   SESSION_STATS   pri_sess_stats;     /* primary hop session stats    */
   SESSION_STATS   sec_sess_stats;     /* secondary hop session        */
                                       /* statistics                   */
   unsigned char   reserva[20];        /* reserved                     */
} ISR_INDICATION;

typedef struct fqpcid
{
   unsigned char   pcid[8];            /* pro correlator identifier    */
   unsigned char   fqcp_name[17];      /* orig's network qualified     */
                                       /* CP name                      */
   unsigned char   reserve3[3];        /* reserved                     */
} FQPCID;

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;   /* Max send pacing window size  */
   unsigned short  cur_send_pac_win;   /* Curr send pacing window size */
   unsigned short  max_rcv_pac_win;    /* Max receive pacing win size  */
   unsigned short  cur_rcv_pac_win;    /* Curr rec pacing window size  */
   unsigned long   send_data_frames;   /* Number of data frames sent   */
   unsigned long   send_fmd_data_frames;
                                       /* num of FMD data frames sent  */
   unsigned long   send_data_bytes;    /* Number of data bytes sent    */
   unsigned long   rcv_data_frames;    /* Num data frames received     */
   unsigned long   rcv_fmd_data_frames;
                                       /* num of FMD data frames recvd */
   unsigned long   rcv_data_bytes;     /* Num 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[8];        /* Link station name            */
   unsigned char   pacing_type;       /* type of pacing in use        */
} SESSION_STATS;
```

Parameters

The application supplies the following parameters:

opcode
AP_ISR_INDICATION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure which has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields may be set to null. The application should issue a QUERY verb to update the information which has been lost.

deactivated
Set to AP_YES when the ISR session is deactivated. Set to AP_NO when the session is activated.

fqpcid.pcid
Procedure Correlator ID. This is an 8-byte hexadecimal string.

fqpcid.fqcp_name
Fully qualified Control Point name. This name is 17-bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

fqplu_name
Fully qualified primary LU name (as specified on the BIND request). This name is 17-bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This name will be all zeros if **deactivated** is AP_YES.

fqslu_name
Fully qualified secondary LU name (as specified on the BIND request). This name is 17-bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.) This name will be all zeros if **deactivated** is AP_YES.

mode_name
Mode name which designates the network properties for a group of sessions (as specified on the BIND request). This is an 8–byte alphanumeric type A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This name will be all zeros if deactivated is AP_YES.

cos_name
Class of Service name. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces. This name will be all zeros if **deactivated** is AP_YES.

transmission_priority
The transmission priority associated with the session. This field is reserved if **deactivated** is AP_YES.

sense_data
The sense data sent or received on the UNBIND request. This field is reserved if **deactivated** is AP_YES.

pri_sess_stats.rcv_ru_size
Maximum receive RU size.

pri_sess_stats.send_ru_size
Maximum send RU size.

pri_sess_stats.max_send_btu_size
Maximum BTU size that can be sent.

pri_sess_stats.max_rcv_btu_size
Maximum BTU size that can be received.

pri_sess_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

pri_sess_stats.cur_send_pac_win
Current size of the send pacing window on this session.

pri_sess_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

pri_sess_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

pri_sess_stats.send_data_frames
Number of normal flow data frames sent.

pri_sess_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

pri_sess_stats.send_data_bytes
Number of normal flow data bytes sent.

pri_sess_stats.rcv_data_frames
Number of normal flow data frames received.

pri_sess_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

pri_sess_stats.rcv_data_bytes
Number of normal flow data bytes received.

pri_sess_stats.sidh
Session ID high byte.

pri_sess_stats.sidl
Session ID low byte.

pri_sess_stats.odai
Origin destination address 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.

pri_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.

pri_sess_stats.pacing_type
Receive pacing type in use on the primary session. This can take the values AP_NONE, AP_PACING_FIXED, or AP_PACING_ADAPTIVE .

sec_sess_stats.rcv_ru_size
Maximum receive RU size.

sec_sess_stats.send_ru_size
Maximum send RU size.

sec_sess_stats.max_send_btu_size
Maximum BTU size that can be sent.

sec_sess_stats.max_rcv_btu_size
Maximum BTU size that can be received.

sec_sess_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

sec_sess_stats.cur_send_pac_win
Current size of the send pacing window on this session.

sec_sess_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

sec_sess_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

sec_sess_stats.send_data_frames
Number of normal flow data frames sent.

sec_sess_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

sec_sess_stats.send_data_bytes
Number of normal flow data bytes sent.

sec_sess_stats.rcv_data_frames
Number of normal flow data frames received.

sec_sess_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

sec_sess_stats.rcv_data_bytes
Number of normal flow data bytes received.

sec_sess_stats.sidh
Session ID high byte.

sec_sess_stats.sidl
Session ID low byte.

sec_sess_stats.odai
Origin destination address 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.

sec_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.

sec_sess_stats.pacing_type
Receive pacing type in use on the secondary session. This can take the values AP_NONE, AP_PACING_FIXED, or AP_PACING_ADAPTIVE .

LOCAL_LU_INDICATION

This indication is generated whenever a LOCAL LU is defined or deleted. This allows a registered application to maintain a list of all local LUs currently defined.

VCB Structure

```
typedef struct local_lu_indication
{
   unsigned short  opcode;             /* verb operation code            */
   unsigned char   reserv2;            /* reserved                       */
   unsigned char   format;             /* format                         */
   unsigned short  primary_rc;         /* primary return code            */
   unsigned long   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   description[RD_LEN];
                                       /* 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
{
   unsigned short  rcv_ru_size;       /* session receive RU size        */
   unsigned short  send_ru_size;      /* session send RU size           */
   unsigned short  max_send_btu_size; /* max send BTU size              */
   unsigned short  max_rcv_btu_size;  /* max rcv BTU size               */
   unsigned short  max_send_pac_win;  /* max send pacing window size    */
   unsigned short  cur_send_pac_win;  /* current send pacing win size   */
   unsigned short  max_rcv_pac_win;   /* max receive pacing win size    */
   unsigned short  cur_rcv_pac_win;   /* curr receive pacing winsize    */
   unsigned long   send_data_frames;  /* number of data frames sent     */
   unsigned long   send_fmd_data_frames;
                                      /* num of FMD data frames sent    */
   unsigned long   send_data_bytes;   /* number of data bytes sent      */
   unsigned long   rcv_data_frames;   /* num 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           */
   unsigned char   sidl;              /* session ID low byte            */
   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;
```

Note: The LU-SSCP statistics are only valid when both **nau_address** is nonzero and the LU-SSCP session goes from active to inactive. In all other cases the fields are reserved.

Parameters

opcode
AP_LOCAL_LU_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES, then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

reason Reason for indication being issued:

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 LU. Name of the local LU whose state has changed. This is an 8-byte alphanumeric type A EBCDIC string (starting with a letter), padded to 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 8 bytes are significant.

nau_address
Network addressable unit address of the LU, which must be in the range 0–255. A nonzero value implies the LU is a dependent LU. Zero implies the LU is an independent LU.

pu_name
Name of the PU that this LU uses. This is an 8-byte alphanumeric type A EBCDIC string. This field is only significant if the LU is a dependent LU (that is, **nau_address** is nonzero), and will be set to all binary zeros for independent LUs.

lu_sscp_sess_active
Specifies whether the LU-SSCP session is active (AP_YES or AP_NO). If **nau_address** is zero then this field is reserved.

lu_sscp_stats.rcv_ru_size
This field is always reserved.

lu_sscp_stats.send_ru_size
This field is always reserved.

lu_sscp_stats.max_send_btu_size
Maximum BTU size that can be sent.

lu_sscp_stats.max_rcv_btu_size
Maximum BTU size that can be received.

lu_sscp_stats.max_send_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_rcv_pac_win
This field will always be 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 FMD data 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 FMD data 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 address indicator. When bringing up a session, the sender of the ACTLU sets this field to zero if the local node contains the primary link station, and sets it to 1 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 8 bytes are significant. This field can be used to correlate this session with the link over which the session flows.

lu_sscp_stats.pacing_type
Receiving pacing type in use on the LU-SSCP session. This will take the value AP_NONE.

sscp_id
This is a 6–byte field containing the SSCP ID received in the ACTPU for the PU used by this LU.

This field is only used by dependent LUs, and will be set to all binary zeros for independent LUs or if **lu_sscp_sess_active** is not set to AP_YES.

LOCAL_TOPOLOGY_INDICATION

This indication is generated when a TG entry in a node's local topology database changes from active to inactive, or from inactive to active.

VCB Structure

```
typedef struct local_topology_indication
{
   unsigned short  opcode;         /* verb operation code           */
   unsigned char   reserv2;        /* reserved                      */
   unsigned char   format;         /* format                        */
   unsigned short  primary_rc;     /* primary return code           */
   unsigned long   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   cp_cp_session_active;
                                   /* CP-CP session is active       */
   unsigned char   branch_link_type;
                                   /* branch link type              */
   unsigned char   branch_tg;      /* TG is a branch TG             */
   unsigned char   reserva[17];    /* reserved                      */
} LOCAL_TOPOLOGY_INDICATION;
```

Parameters

opcode
AP_LOCAL_TOPOLOGY_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

status Specifies the status of the TG. This can be one or more of the following values ORed together:

AP_TG_OPERATIVE
AP_TG_CP_CP_SESSIONS
AP_TG_QUIESCING
AP_NONE

dest Fully qualified destination node name for the TG. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

dest_type
Type of the node. It is one of the following values:

AP_END_NODE
AP_NETWORK_NODE
AP_VRN

tg_num
Number associated with the TG.

cp_cp_session_active
Specifies whether the local node's contention winner CP-CP session is active (AP_NO or AP_YES).

branch_link_type
BrNN only. This branch link type of this TG. This is set to one of the following:

AP_UPLINK
This link is an uplink.

AP_DOWNLINK
The link is a downlink to an EN.

AP_DOWNLINK_TO_BRNN
The TG is a downlink to a BrNN that is showing its EN face.

AP_OTHERLINK
This link is an otherlink.

Other node types: This field is not meaningful and is always set to AP_BRNN_NOT_SUPPORTED.

branch_tg
NN only. Sepcifies whether the TG is a branch TG.

AP_NO
The TG is not a branch TG.

AP_YES
The TG is a branch TG.

Other node types: This field is not meaningful and is always set to AP_NO.

LS_INDICATION

This indication is generated when the number of active sessions using the link changes, or the external state of the link station changes. Link station statistics are supplied when the link station becomes inactive.

VCB Structure

```
typedef struct ls_indication
{
   unsigned short  opcode;             /* verb operation code          */
   unsigned char   attributes;         /* verb attributes              */
   unsigned char   reserv2;            /* reserved                     */
   unsigned char   format;             /* format                       */
   unsigned short  primary_rc;         /* primary return code          */
   unsigned long   secondary_rc;       /* secondary return code        */
   unsigned char   data_lost;          /* previous indication lost     */
   unsigned char   deactivated;        /* has session been deactivated? */
   unsigned char   ls_name[8];         /* link station name            */
   unsigned char   description[RD_LEN]; /* resource description        */
   unsigned char   adj_cp_name[17];    /* network qualified Adj CP name */
   unsigned char   adj_node_type;      /* adjacent node type           */
   unsigned short  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                    */
   unsigned long   sense_data;         /* sense data                   */
   unsigned char   brnn_link_type;     /* branch link type             */
   unsigned char   adj_cp_is_brnn;     /* adjacent CP is a BrNN        */
   unsigned char   reserva[17];        /* reserved                     */
} LS_INDICATION;
```

```
typedef struct ls_stats
{
   unsigned long   in_xid_bytes;       /* num of XID bytes received    */
   unsigned long   in_msg_bytes;       /* num message bytes received   */
   unsigned long   in_xid_frames;      /* num XID frames received      */
   unsigned long   in_msg_frames;      /* num message frames received  */
   unsigned long   out_xid_bytes;      /* num XID bytes sent           */
   unsigned long   out_msg_bytes;      /* num message bytes sent       */
   unsigned long   out_xid_frames;     /* number of XID frames sent    */
   unsigned long   out_msg_frames;     /* num message frames sent      */
   unsigned long   in_invalid_sna_frames;
                                       /* num invalid frames recvd     */
   unsigned long   in_session_control_frames;
                                       /* number of control            */
                                       /* frames recvd                 */
   unsigned long   out_session_control_frames;
                                       /* number of control            */
                                       /* frames sent                  */
   unsigned long   echo_rsps;          /* response from adj LS count   */
   unsigned long   current_delay;      /* time taken for last          */
                                       /* test signal                  */
   unsigned long   max_delay;          /* max delay by test signal     */
   unsigned long   min_delay;          /* min delay by test signal     */
   unsigned long   max_delay_time;     /* time since longest delay     */
   unsigned long   good_xids;          /* successful XID on LS count   */
   unsigned long   bad_xids;           /* unsuccessful XID on LS count */
} LS_STATS;
```

Parameters

opcode
AP_LS_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_YES when the LS becomes inactive. Set to AP_NO when the LS becomes active.

ls_name
Name of link station. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

description
Resource description (as specified on DEFINE_LS). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

adj_cp_name
Fully-qualified, 17-byte long, adjacent control point name. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

adj_node_type
Type of the node. It is one of the following values:

AP_END_NODE
AP_NETWORK_NODE
AP_LEN_NODE
AP_VRN

act_sess_count
Total number of active sessions (both endpoint and intermediate) using the link.

indication_cause
Cause of the indication. It is one of the following values:

AP_ACTIVATION_STARTED
The link is activating.

AP_ACTIVATING
The link has become active.

AP_DEACTIVATION_STARTED
The link is being deactivated.

AP_DEACTIVATING
The link has become inactive.

AP_SESS_COUNT_CHANGING
The number of active sessions using the link has changed.

AP_CP_NAME_CHANGING
An adjacent node has changed its control point name.

AP_FAILED
The link has failed.

AP_ACTIVATION_FAILED
The link failed to activate.

AP_PENDING_RETRY
A retry timer has been started. When the timer expires, activation of the link is automatically retried.

AP_DATA_LOST
A previous indication has been lost. Note that link station statistics are only supplied when the link station goes from active to inactive (that is, deactivating is set to AP_YES and **indication_cause** is set to AP_DEACTIVATING). In all other cases the fields are reserved.

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 incorrect frames 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.echo_rsps
Number of echo responses received from the adjacent node. Echo requests are sent periodically to gauge the propagation delay to the adjacent node.

ls_stats.current_delay
Time (in milliseconds) that it took for the last test signal to be sent and returned from this link station to the adjacent link station.

ls_stats.max_delay
Longest time taken (in milliseconds) for a test signal to be sent and returned from this link station to the adjacent link station.

ls_stats.min_delay
Shortest time taken (in milliseconds) for a test signal to be sent and returned from this link station to the adjacent link station.

ls_stats.max_delay_time
Time since system startup (in hundredths of a second) when the longest delay occurred.

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
Number associated with the TG.

sense_data
This sense data is set if Personal Communications or Communications Server detects an XID protocol error. This field is reserved unless **indication_cause** is AP_FAILED.

brnn_link_type
BrNN only. This branch link type. It is one of the following:

AP_UPLINK
This link is an uplink.

AP_DOWNLINK
The link is a downlink.

AP_OTHERLINK
This link is an otherlink.

AP_UNKNOWN_LINK_TYPE
This link is an otherlink.

Other node types: This field is not meaningful and is always set to AP_BRNN_NOT_SUPPORTED.

adj_cp_is_brnn
All node types: Specifies whether the adjacent node is a BrNN.

AP_UNKNOWN
It is not known whether the adjacent node is a BrNN.

AP_NO
The adjacent node is not a BrNN.

AP_YES
The adjacent node is BrNN.

LU_0_TO_3_INDICATION

This indication is generated when the state of a local LU (Type 0-3) changes.

VCB Structure

```
typedef struct lu_0_to_3_indication
{
   unsigned short  opcode;             /* verb operation code          */
   unsigned char   attributes;         /* attributes                   */
   unsigned char   reserv2;            /* reserved                     */
   unsigned char   format;             /* format                       */
   unsigned short  primary_rc;         /* primary return code          */
   unsigned long   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[RD_LEN]; /* resource description        */
   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-SLU session statistics   */
   unsigned char   sscp_id[16];        /* SSCP ID                      */
} LU_0_TO_3_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;  /* max send BTU size            */
   unsigned short  max_rcv_btu_size;   /* max rcv BTU size             */
   unsigned short  max_send_pac_win;   /* max send pacing window size  */
   unsigned short  cur_send_pac_win;   /* current send pacing win size */
   unsigned short  max_rcv_pac_win;    /* max receive pacing win size  */
   unsigned short  cur_rcv_pac_win;    /* curr receive pacing winsize  */
   unsigned long   send_data_frames;   /* number of data frames sent   */
   unsigned long   send_fmd_data_frames;
                                       /* num of FMD data frames sent  */
   unsigned long   send_data_bytes;    /* number of data bytes sent    */
   unsigned long   rcv_data_frames;    /* num 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         */
   unsigned char   sidl;               /* session ID low byte          */
   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;
```

Parameters

opcode
AP_LU_0_TO_3_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

If data lost is set to AP_YES, this is set to AP_EXTERNALLY_VISIBLE.

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

pu_name
Name of local PU. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

lu_name
Name of the local LU whose state has changed. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
Resource description (as specified on DEFINE_LU_0_TO_3). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

nau_address
Network addressable unit address of the LU (which will be in the range 10–2554).

lu_sscp_sess_active
Specifies whether the ACTLU has been successfully processed (AP_YES or AP_NO).

appl_conn_active
Set if the application is using this LU (AP_YES or AP_NO).

plu_sess_active
Specifies whether the PLU-SLU session has been activated (AP_YES or AP_NO).

host_attachment
Specifies the LU host attachment type:

AP_DLUR_ATTACHED
LU is attached to host system using DLUR.

AP_DIRECT_ATTACHED
LU is directly attached to host system. Note the LU-SSCP and PLU-SLU statistics are only valid when the sessions go from active to inactive. In all other cases the fields are reserved.

lu_sscp_stats.rcv_ru_size
This field is always reserved.

lu_sscp_stats.send_ru_size
This field is always reserved.

lu_sscp_stats.max_send_btu_size
Maximum BTU size that can be sent.

lu_sscp_stats.max_rcv_btu_size
Maximum BTU size that can be received.

lu_sscp_stats.max_send_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_send_pac_win
This field will always be set to zero.

lu_sscp_stats.max_rcv_pac_win
This field will always be set to zero.

lu_sscp_stats.cur_rcv_pac_win
This field will always be 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 FMD data 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 FMD data 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 address indicator. When bringing up a session, the sender of the ACTLU sets this field to zero if the local node contains the primary link station, and sets it to 1 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 8 bytes are significant. This field can be used to correlate this session with the link over which the session flows.

lu_sscp_stats.pacing_type
Receiving pacing type in use on the LU-SSCP session. This will take the value AP_NONE.

plu_stats.rcv_ru_size
Maximum receive RU size.

plu_stats.send_ru_size
Maximum send RU size.

plu_stats.max_send_btu_size
Maximum BTU size that can be sent.

plu_stats.max_rcv_btu_size
Maximum BTU size that can be received.

plu_stats.max_send_pac_win
Maximum size of the send pacing window on this session.

plu_stats.cur_send_pac_win
Current size of the send pacing window on this session.

plu_stats.max_rcv_pac_win
Maximum size of the receive pacing window on this session.

plu_stats.cur_rcv_pac_win
Current size of the receive pacing window on this session.

plu_stats.send_data_frames
Number of normal flow data frames sent.

plu_stats.send_fmd_data_frames
Number of normal flow FMD data frames sent.

plu_stats.send_data_bytes
Number of normal flow data bytes sent.

plu_stats.rcv_data_frames
Number of normal flow data frames received.

plu_stats.rcv_fmd_data_frames
Number of normal flow FMD data frames received.

plu_stats.rcv_data_bytes
Number of normal flow data bytes received.

plu_stats.sidh
Session ID high byte.

plu_stats.sidl
Session ID low byte.

plu_stats.odai
Origin destination address indicator. When bringing up a session, the sender of the ACTLU sets this field to zero if the local node contains the primary link station, and sets it to 1 if the ACTLU sender is the node containing the secondary link station.

plu_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 this session with the link over which the session flows.

plu_stats.pacing_type
Receiving pacing type in use on the PLU-SLU session. This will take the value AP_NONE or AP_PACING_FIXED.

sscp_id
This is 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_YES, then this field will be zeroed.

MODE_INDICATION

This indication is sent when a local LU and partner LU combination start to use a particular mode, and when the current session count for the local LU, partner LU, and mode combination changes.

VCB Structure

```
typedef struct mode_indication
{
   unsigned short  opcode;               /* verb operation code       */
   unsigned char   reserv2;              /* reserved                  */
   unsigned char   format;               /* format                    */
   unsigned short  primary_rc;           /* primary return code       */
   unsigned long   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[RD_LEN];  /* resource description      */
   unsigned short  curr_sess_count;      /* current session count     */
   unsigned char   reserva[20];          /* reserved                  */
} MODE_INDICATION;
```

Parameters

opcode
AP_MODE_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

removed
Specifies whether an entry is being removed (AP_YES or AP_NO). It is set when entry is being removed rather than added.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is

composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
Resource description (as specified on DEFINE_MODE). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

curr_sess_count
Current count of sessions for this local LU, partner LU, and mode combination.

NN_TOPOLOGY_NODE_INDICATION

This verb applies only to Communications Server.

This indication is generated when a node entry in a network node's topology database changes from active to inactive, or from inactive to active.

VCB Structure

```
typedef struct nn_topology_node_indication
{
   unsigned short  opcode;         /* verb operation code           */
   unsigned char   reserv2;        /* reserved                      */
   unsigned char   format;         /* format                        */
   unsigned short  primary_rc;     /* primary return code           */
   unsigned long   secondary_rc;   /* secondary return code         */
   unsigned char   data_lost;      /* previous indication lost      */
   unsigned char   deactivated;    /* has the node become inactive? */
   unsigned char   node_name[17];  /* node name                     */
   unsigned char   node_type;      /* node type                     */
   unsigned char   branch_aware;   /* node is branch aware          */
   unsigned char   reserva[19];    /* reserved                      */
} NN_TOPOLOGY_NODE_INDICATION;
```

Parameters

opcode
AP_NN_TOPOLOGY_NODE_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_YES when the node becomes inactive. Set to AP_NO when the node becomes active.

node_name
Network qualified node name from network topology database. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

node_type
Type of the node. It is one of the following.

AP_NETWORK_NODE
AP_VRN

branch_aware
Specifies whether the node is branch aware.

AP_NO
The node is not branch aware.

AP_YES
The node is branch aware.

NN_TOPOLOGY_TG_INDICATION

This verb applies only to Communications Server.

This indication is generated when a TG entry in a network node's topology database changes from active to inactive, or from inactive to active.

VCB Structure

```
typedef struct nn_topology_tg_indication
{
   unsigned short  opcode;         /* verb operation code           */
   unsigned char   reserv2;        /* reserved                      */
   unsigned char   format;         /* format                        */
   unsigned short  primary_rc;     /* primary return code           */
   unsigned long   secondary_rc;   /* secondary return code         */
   unsigned char   data_lost;      /* previous indication lost      */
   unsigned char   status;         /* TG status                     */
   unsigned char   owner[17];      /* name of TG owner node         */
   unsigned char   dest[17];       /* name of TG destination node   */
   unsigned char   tg_num;         /* TG number                     */
   unsigned char   owner_type;     /* Type of node that owns the TG */
   unsigned char   dest_type;      /* TG destination node type      */
   unsigned char   cp_cp_session_active;
                                   /* CP-CP session is active       */
   unsigned char   branch_tg;      /* TG is a branch TG             */
   unsigned char   reserva[16];    /* reserved                      */
} NN_TOPOLOGY_TG_INDICATION;
```

Parameters

opcode
AP_NN_TOPOLOGY_TG_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

status
Specifies the status of the TG. This can be one or more of the following values ORed together:

AP_TG_OPERATIVE
AP_TG_QUIESCING
AP_TG_CP_CP_SESSIONS
AP_NONE

owner
Name of the TG's originating node (always set to the local node name). This name is 17 bytes long and is right-padded with EBCDIC spaces. It is

composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

dest Fully qualified destination node name for the TG. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

tg_num
Number associated with the TG.

owner_type
Type of the node that owns the TG.

AP_NETWORK_NODE
AP_VRN

dest_type
Type of the node.

AP_NETWORK_NODE
AP_VRN

cp_cp_session_active
Specifies whether the owning node's contention winner CP-CP session is active (AP_NO or AP_YES).

branch_tg
Sepcifies whether the TG is a branch TG.

AP_NO
The TG is not a branch TG.

AP_YES
The TG is a branch TG.

PLU_INDICATION

This indication is generated when a local LU first connects to a partner LU. This will happen when the first ALLOCATE to this PLU is processed or when the first BIND is received from this PLU. This indication is also generated if the partner control point name changes.

VCB Structure

```
typedef struct plu_indication
{
   unsigned short  opcode;               /* verb operation code       */
   unsigned char   reserv2;              /* reserved                  */
   unsigned char   format;               /* format                    */
   unsigned short  primary_rc;           /* primary return code       */
   unsigned long   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[RD_LEN];  /* resource description      */
   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

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether one or more indications have been lost (AP_YES or AP_NO). It is set when an internal component was unable to send a previous indication. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

removed
Specifies whether an entry is being removed (AP_YES or AP_NO). It is set when entry is being removed rather than added.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

description
Resource description (as specified on DEFINE_PARTNER_LU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

partner_cp_name
17-byte fully qualified network name for the control point of the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

partner_lu_located
Specifies whether the partner control point name has been resolved (AP_YES or AP_NO), and thus whether the **partner_cp_name** field contains the control point name.

PORT_INDICATION

This indication is generated when the port goes from active to inactive (or vice versa).

VCB Structure

```
typedef struct port_indication
{
   unsigned short  opcode;               /* verb operation code          */
   unsigned char   reserv2;              /* reserved                     */
   unsigned char   attributes;           /* verb attributes              */
   unsigned char   format;               /* format                       */
   unsigned short  primary_rc;           /* primary return code          */
   unsigned long   secondary_rc;         /* secondary return code        */
   unsigned char   data_lost;            /* previous indication lost     */
   unsigned char   deactivated;          /* has session been deactivated? */
   unsigned char   port_name[8];         /* link station name            */
   unsigned char   description[RD_LEN];  /* resource description         */
   unsigned char   reserva[20];          /* reserved                     */
} PORT_INDICATION;
```

Parameters

opcode
AP_PORT_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_YES when the port becomes inactive. Set to AP_NO when the port becomes active.

port_name
Name of port. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

description
Resource description (as specified on DEFINE_PORT). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

PU_INDICATION

This indication is generated when the state of a local PU changes.

VCB Structure

```
typedef struct pu_indication
{
   unsigned short  opcode;               /* verb operation code          */
   unsigned char   attributes;           /* attributes                   */
   unsigned char   reserv2;              /* reserved                     */
   unsigned char   format;               /* format                       */
   unsigned short  primary_rc;           /* primary return code          */
   unsigned long   secondary_rc;         /* secondary return code        */
   unsigned char   data_lost;            /* previous indication lost     */
   unsigned char   pu_name[8];           /* PU Name                      */
   unsigned char   description[RD_LEN];  /* resource description         */
   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   */
   unsigned char   sscp_id[6];           /* SSCP ID                      */
} 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;    /* max send BTU size            */
   unsigned short  max_rcv_btu_size;     /* max rcv BTU size             */
   unsigned short  max_send_pac_win;     /* max send pacing window size  */
   unsigned short  cur_send_pac_win;     /* curr send pacing window size */
   unsigned short  max_rcv_pac_win;      /* max rcv pacing window size   */
   unsigned short  cur_rcv_pac_win;      /* curr receive pacing win 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;      /* num of data frames received  */
   unsigned long   rcv_fmd_data_frames;
                                         /* num FMD data frames received */
   unsigned long   rcv_data_bytes;       /* num 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[8];           /* Link station name            */
   unsigned char   pacing_type;          /* type of pacing in use        */
} SESSION_STATS;
```

Parameters

opcode
AP_PU_INDICATION

attributes
The attributes of the verb. This field is a bit field. The first bit contains the visibility of the resource to be defined and corresponds to one of the following:

AP_EXTERNALLY_VISIBLE
AP_INTERNALLY_VISIBLE

If data lost is set to AP_YES, this is set to AP_EXTERNALLY_VISIBLE.

format
: Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
: AP_OK

secondary_rc
: Equals zero.

data_lost
: Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

pu_name
: Name of the PU (configured on the DEFINE_LS verb). This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

description
: Resource description (as specified on DEFINE_LS or DEFINE_INTERNAL_PU). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

pu_sscp_sess_active
: Specifies whether the ACTPU has been successfully processed (AP_YES or AP_NO).

host_attachment
: PU host attachment type:

  AP_DLUR_ATTACHED
  : PU is attached to host system using DLUR.

  AP_DIRECT_ATTACHED
  : PU is directly attached to host system.

    Note: PU-SSCP statistics are valid only when the session state has moved from active to inactive.

    In all other cases the following fields are reserved:

pu_sscp_stats.rcv_ru_size
: This field is always reserved.

pu_sscp_stats.send_ru_size
: This field is always reserved.

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
: This field will always be set to zero.

pu_sscp_stats.cur_send_pac_win
: This field will always be set to zero.

pu_sscp_stats.max_rcv_pac_win
: This field will always be set to zero.

pu_sscp_stats.cur_rcv_pac_win
This field will always be 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 address indicator. When bringing up a session, the sender of the ACTPU sets this field to zero if the local node contains the primary link station, and sets it to 1 if the ACTPU 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 string in a locally displayable character set. All 8 bytes are significant. This field can be used to correlate this session with the link over which the session flows.

pu_stats.pacing_type
Receiving pacing type in use on the PU-SSCP session. This will take the value AP_NONE.

sscp_id
This is a 6–byte field containing the SSCP ID received in the ACTPU for this PU.

If **plu_sscp_sess_active** is not AP_YES, then this field will be zeroed.

REGISTRATION_FAILURE

REGISTRATION_FAILURE indicates that an attempt to register resources with the network node server failed.

VCB Structure

```
typedef struct registration_failure
{
   unsigned short  opcode;               /* verb operation code      */
   unsigned char   reserv2;              /* reserved                 */
   unsigned char   format;               /* format                   */
   unsigned short  primary_rc;           /* primary return code      */
   unsigned long   secondary_rc;         /* secondary return code    */
   unsigned char   data_lost;            /* previous indication lost */
   unsigned char   resource_name[17];    /* network qualified        */
                                         /* resource name            */
   unsigned short  resource_type;        /* resource type            */
   unsigned char   description[RD_LEN];  /* resource description     */
   unsigned char   reserv2b[2];          /* reserved                 */
   unsigned long   sense_data;           /* sense data               */
   unsigned char   reserva[20];          /* reserved                 */
} REGISTRATION_FAILURE;
```

Parameters

opcode
AP_REGISTRATION_FAILURE

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES, then subsequent data fields may be set to null. The application should issue a QUERY verb to update the information that has been lost.

resource_name
Name of resource that failed to register. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

resource_type
Resource type. One of the following values:

AP_NNCP_RESOURCE
AP_ENCP_RESOURCE
AP_LU_RESOURCE

description
Resource description (as specified on DEFINE_LOCAL_LU, or DEFINE_ADJACENT_NODE).

sense_data
Sense data (specified in *SNA Formats*).

RTP_INDICATION

This indication is generated when:

- An RTP connection is connected or disconnected
- The active session count changes
- The connection performs a path-switch

When the connection is disconnected, final RTP statistics will be returned. At other times the **rtp_stats** field is reserved.

VCB Structure

```
typedef struct rtp_indication
{
   unsigned short  opcode;             /* verb operation code           */
   unsigned char   reserv2;            /* reserved                      */
   unsigned char   format;             /* format                        */
   unsigned short  primary_rc;         /* primary return code           */
   unsigned long   secondary_rc;       /* secondary return code         */
   unsigned char   data_lost;          /* previous indication(s) lost   */
   unsigned char   connection_state;   /* the current state of the RTP  */
                                       /* connection                    */
   unsigned char   rtp_name[8];        /* name of the RTP connection    */
   unsigned short  num_sess_active;    /* number of active sessions     */
   unsigned char   indication_cause;   /* reason for this indication    */
   unsigned char   reserv3[3];         /* reserved                      */
   RTP_STATISTICS  rtp_stats;          /* RTP statistics                */
} RTP_INDICATION;
```

```
 typedef struct rtp_statistics
{
   unsigned long   bytes_sent;         /* total num of bytes sent       */
   unsigned long   bytes_received;     /* total num bytes received      */
   unsigned long   bytes_resent;       /* total num of bytes resent     */
   unsigned long   bytes_discarded;    /* total num bytes discarded     */
   unsigned long   packets_sent;       /* total num of packets sent     */
   unsigned long   packets_received;   /* total num packets received    */
   unsigned long   packets_resent;     /* total num of packets resent   */
   unsigned long   packets_discarded;  /* total num packets discarded   */
   unsigned long   gaps_detected;      /* gaps detected                 */
   unsigned long   send_rate;          /* current send rate             */
   unsigned long   max_send_rate;      /* maximum send rate             */
   unsigned long   min_send_rate;      /* minimum send rate             */
   unsigned long   receive_rate;       /* current receive rate          */
   unsigned long   max_receive_rate;   /* maximum receive rate          */
   unsigned long   min_receive_rate;   /* minimum receive rate          */
   unsigned long   burst_size;         /* current burst size            */
   unsigned long   up_time;            /* total uptime of connection    */
   unsigned long   smooth_rtt;         /* smoothed round-trip time      */
   unsigned long   last_rtt;           /* last round-trip time          */
   unsigned long   short_req_timer;    /* SHORT_REQ timer duration      */
   unsigned long   short_req_timeouts; /* number of SHORT_REQ timeouts  */
   unsigned long   liveness_timeouts;  /* number of liveness timeouts   */
   unsigned long   in_invalid_sna_frames;
                                       /* number of invalid SNA frames  */
                                       /* received                      */
   unsigned long   in_sc_frames;       /* number of SC frames received  */
   unsigned long   out_sc_frames;      /* number of SC frames sent      */
   unsigned char   reserve[40];        /* reserved                      */
} RTP_STATISTICS;
```

Parameters

opcode
AP_RTP_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then the data contained might have changed more than once since the previous indication received.

connection_state
The current state of the RTP connection. It is one of the following values:

AP_CONNECTING
Connection setup has started, but is not yet complete.

AP_CONNECTED
The connection is fully active.

AP_DISCONNECTED
The connection is no longer active.

rtp_name
RTP connection name. This name is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

num_sess_active
Number of sessions currently active on the connection.

indication_cause
Cause of the indication. It is one of the following values:

AP_ACTIVATED
The connection has become active.

AP_DEACTIVATED
The connection has become inactive.

AP_PATH_SWITCHED
The connection has successfully completed a path switch.

AP_SESS_COUNT_CHANGING
The number of active sessions using the connection has changed.

AP_SETUP_FAILED
The connection has failed before becoming fully active. Note that RTP connection statistics are only supplied when the connection becomes inactive, that is when **indication_cause** is AP_DEACTIVATED or AP_SETUP_FAILED. In all other cases the fields are reserved.

rtp_stats.bytes_sent
Total number of bytes that the local node has sent on this RTP connection.

rtp_stats.bytes_received
Total number of bytes that the local node has received on this RTP connection.

rtp_stats.bytes_resent
Total number of bytes resent by the local node owing to loss in transit.

rtp_stats.bytes_discarded
Total number of bytes sent by the other end of the RTP connection that were discarded as duplicates of data already received.

rtp_stats.packets_sent
Total number of packets that the local node has sent on this RTP connection.

rtp_stats.packets_received
Total number of packets that the local node has received on this RTP connection.

rtp_stats.packets_resent
Total number of packets resent by the local node owing to loss in transit.

rtp_stats.packets_discarded
Total number of packets sent by the other end of the RTP connection that were discarded as duplicates of data already received.

rtp_stats.gaps_detected
Total number of gaps detected by the local node. Each gap corresponds to one or more lost frames.

rtp_stats.send_rate
Current send rate on this RTP connection (measured in kilobits per second). This is the maximum allowed send rate as calculated by the ARB algorithm.

rtp_stats.max_send_rate
Maximum send rate on this RTP connection (measured in kilobits per second).

rtp_stats.min_send_rate
Minimum send rate on this RTP connection (measured in kilobits per second).

rtp_stats.receive_rate
Current receive rate on this RTP connection (measured in kilobits per second). This is the actual receive rate calculated over the last measurement interval.

rtp_stats.max_receive_rate
Maximum receive rate on this RTP connection (measured in kilobits per second).

rtp_stats.min_receive_rate
Minimum receive rate on this RTP connection (measured in kilobits per second).

rtp_stats.burst_size
Current burst-size on the RTP Connection measured in bytes.

rtp_stats.up_time
Total number of seconds the RTP connection has been active.

rtp_stats.smooth_rtt
Smoothed measure of round-trip time between the local node and the partner RTP node (measured in milliseconds).

rtp_stats.last_rtt
The last measured round-trip time between the local node and the partner RTP node (measured in milliseconds).

rtp_stats.short_req_timer
The current duration used for the SHORT_REQ timer (measured in milliseconds).

rtp_stats.short_req_timeouts
Total number of times the SHORT_REQ timer has expired for this RTP connection.

rtp_stats.liveness_timeouts
Total number of times the liveness timer has expired for this RTP connection. The liveness timer expires when the connection has been idle for the period specified in **rtp_connection_detail.liveness_timer**.

rtp_stats.in_invalid_sna_frames
Total number of SNA frames received and discarded as not valid on this RTP connection.

rtp_stats.in_sc_frames
Total number of session control frames received on this RTP connection.

rtp_stats.out_sc_frames
Total number of session control frames sent on this RTP connection.

SESSION_FAILURE_INDICATION

This indication is generated whenever a session is deactivated. This indication is guaranteed; that is, generated without fail.

VCB Structure

```
typedef struct session_failure_indication
{
   unsigned short  opcode;              /* verb operation code          */
   unsigned char   reserv2;             /* reserved                     */
   unsigned char   format;              /* format                       */
   unsigned short  primary_rc;          /* primary return code          */
   unsigned long   secondary_rc;        /* secondary return code        */
   unsigned char   reserv3[3];          /* 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                   */
   unsigned long   sense_data;          /* sense_data                   */
  } SESSION_FAILURE_INDICATION;
```

Parameters

opcode
AP_SESSION_FAILURE_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

session_id
8-byte identifier of the session.

sense_data
The sense data detailing the cause of the session deactivation.

SESSION_INDICATION

This indication is generated when a session is activated or deactivated. When a session is deactivated, final session statistics will be returned. When a session is activated, the **sess_stats** field is reserved.

VCB Structure

```
typedef struct session_indication
{
   unsigned short  opcode;              /* verb operation code          */
   unsigned char   reserv2;             /* reserved                     */
   unsigned char   format;              /* format                       */
   unsigned short  primary_rc;          /* primary return code          */
   unsigned long   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    */
   unsigned long   sense_data;          /* sense_data                   */
   unsigned char   duplex_support;      /* full-duplex support          */
   SESSION_STATS   sess_stats;          /* session statistics           */
   unsigned char   sscp_id[6];          /* SSCP ID of host              */
   unsigned char   plu_slu_comp_lvl;    /* PLU to SLU compression level */
   unsigned char   slu_plu_comp_lvl;    /* SLU to PLU compressionlevel  */
                                        /* correlator 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
{
   unsigned short  rcv_ru_size;         /* session receive RU size      */
   unsigned short  send_ru_size;        /* session send RU size         */
   unsigned short  max_send_btu_size;   /* max send BTU  size           */
   unsigned short  max_rcv_btu_size;    /* max rcv BTU size             */
   unsigned short  max_send_pac_win;    /* max send pacing window size  */
   unsigned short  cur_send_pac_win;    /* curr send pacing window size */
   unsigned short  max_rcv_pac_win;     /* max receive pacing win size  */
   unsigned short  cur_rcv_pac_win;     /* curr receive pacing win 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;     /* num data frames received     */
   unsigned long   rcv_fmd_data_frames;
                                        /* num FMD data frames received */
   unsigned long   rcv_data_bytes;      /* num 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[8];           /* Link station name            */
   unsigned char   pacing_type;          /* type of pacing in use        */
   unsigned char   reserve;              /* reserved                     */
} SESSION_STATS;
```

Parameters

opcode
AP_SESSION_INDICATION

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

primary_rc
AP_OK

secondary_rc
Equals zero.

data_lost
Specifies whether data has been lost (AP_YES or AP_NO). It is set when an internal component detects a failure that has caused a previous indication to be lost. If the **data_lost** flag is set to AP_YES then subsequent data fields can be set to null. The application should issue a QUERY verb to update the information that has been lost.

deactivated
Set to AP_NO when a session is activated. Set to AP_YES when a session is deactivated.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

mode_name
Mode name, which designates the network properties for a group of sessions. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

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 control point name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC

character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

sense_data
The sense data sent or received on the UNBIND request. This field is reserved if **deactivated** is AP_NO.

duplex_support
Returns the conversation duplex support as negotiated on the BIND. This is one of the following values:

AP_HALF_DUPLEX
Only half-duplex conversations are supported.

AP_FULL_DUPLEX
Full-duplex as well as half-duplex conversations are supported.

AP_UNKNOWN
The conversation duplex support is not known because there are no active sessions to the partner LU.

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 address 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 1 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
Receive pacing type in use on this session. This can take the values AP_NONE, AP_PACING_ADAPTVE or AP_PACING_FIXED.

sscp_id
For dependent LU sessions, this field contains the SSCP ID received in the ACTPU from the host for the PU that the local LU is mapped to. For independent LU sessions, this field will be set to all binary zeros.

plu_slu_comp_lvl
Specifies the compression level for data sent from the PLU to the SLU.

AP_NONE
Compression is not used.

AP_RLE_COMPRESSION
RLE compression is used.

AP_LZ9_COMPRESSION
This node can support LZ9 compression.

AP_LZ10_COMPRESSION
The node can support LZ10 compression.

AP_LZ12_COMPRESSION
The node can support LZ12 compression.

slu_plu_comp_lvl
Specifies the compression level for data sent from the SLU to the PLU.

AP_NONE
Compression is not used.

AP_RLE_COMPRESSION
RLE compression is used.

AP_LZ9_COMPRESSION
This node can support LZ9 compression.

AP_LZ10_COMPRESSION
The node can support LZ10 compression.

AP_LZ12_COMPRESSION
The node can support LZ12 compression.

UNREGISTER_INDICATION_SINK

UNREGISTER_INDICATION_SINK removes the identifications of processes and queues that are receiving unsolicited indications.

If the specified combination of **proc_id, queue_id,** and **indication_opcode** has only been registered once, the entry is removed. If the specified combination has been registered more than once, the entry that matches **orig_verb_data**in the verb_signal header of UNREGISTER_INDICATION_SINK is removed.

VCB Structure

```
typedef struct unregister_indication_sink
{
   unsigned short  opcode;              /* verb operation code          */
   unsigned char   reserv2;             /* reserved                     */
   unsigned char   format;              /* format                       */
   unsigned short  primary_rc;          /* primary return code          */
   unsigned long   secondary_rc;        /* secondary return code        */
   unsigned PROC_ID
                   proc_id;             /* process identifier of sink   */
   unsigned QUEUE_ID
                   queue_id;            /* queue identifier where       */
                                        /* indications will be sent     */
   unsigned short  indication_opcode;   /* opcode of indication to      */
                                        /* be sunk                      */
} UNREGISTER_INDICATION_SINK;
```

Parameters

opcode
AP_UNREGISTER_INDICATION_SINK

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

proc_id
Process ID of process where indication are being sent.

queue_id
Queue ID of queue where indications are being sent.

indication_opcode
Opcode of indications that are being returned.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_OP_CODE

AP_DYNAMIC_LOAD_ALREADY_REGD

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_LU_NAME

If the verb does not execute because one or more of the relevant START_NODE parameters were not set, the Program returns the following parameter:

primary_rc
AP_FUNCTION_NOT_SUPPORTED

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because a STOP_NODE verb has been issued, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 10. Security Verbs

This chapter describes verbs used to define and delete security passwords.

CONV_SECURITY_BYPASS

CONV_SECURITY_BYPASS allows an application to control whether the Program will enforce conversation-level security for a local LU. Once security has been bypassed, the Program will not do any authentication or authorization for the conversations on the local LU.

VCB Structure

```
typedef struct conv_security_bypass
{
   unsigned short  opcode;          /* verb operation code       */
   unsigned char   reserv2;         /* reserved                  */
   unsigned char   format;          /* format                    */
   unsigned short  primary_rc;      /* primary return code       */
   unsigned long   secondary_rc;    /* secondary return code     */
   unsigned char   lu_name[8];      /* local LU name             */
   unsigned char   lu_alias[8];     /* local LU alias            */
   unsigned char   bypass_security; /* should security be        */
                                    /* bypassed?                 */
   unsigned char   reserv3[3];      /* reserved                  */
} CONV_SECURITY_BYPASS;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_CONV_SECURITY_BYPASS

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Local LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_alias** and the **lu_name** are set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

bypass_security
Specifies whether security should be bypassed (AP_YES or AP_NO).

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_LU_ALIAS
AP_INVALID_BYPASS_SECURITY

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

CREATE_PASSWORD_SUBSTITUTE

CREATE_PASSWORD_SUBSTITUTE returns the password substitute, password verifier, and the send sequence number used to generate the substitute and verifier for the specified session.

VCB Structure

```
typedef struct create_password_substitute
{
   unsigned short  opcode;           /* verb operation code          */
   unsigned char   reserv2;          /* reserved                     */
   unsigned char   format;           /* format                       */
   unsigned short  primary_rc;       /* primary return code          */
   unsigned long   secondary_rc;     /* secondary return code        */
   unsigned char   lu_alias[8];      /* LU alias                     */
   unsigned char   conv_group_id[8]; /* partner LU alias             */
   unsigned char   user_id[10];      /* user ID                      */
   unsigned char   pw[10];           /* clear text password          */
   unsigned char   seq_no[8];        /* sequence number              */
   unsigned char   pw_sub[10];       /* password substitute          */
   unsigned char   pw_verifier[10];  /* password verifier            */
} CREATE_PASSWORD_SUBSTITUTE;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_CREATE_PASSWORD_SUBSTITUTE

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set.

conv_group_id
Conversation group identifier for the session used by the LU.

user_id
The user ID.

pw Clear text password to be used in the encryption algorithm.

Returned Parameters

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

primary_rc
AP_OK

seq_no
Send sequence number used in the encryption algorithm. Note, if the verb is successful, the internal value of the send sequence number for this session is incremented. The value returned is the value after incrementing.

pw_sub
Password substitute generated by the encryption algorithm.

pw_verifier
Password verifier generated by the encryption algorithm.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_LU_ALIAS

AP_DEACT_CG_INVALID_CGID

If the verb does not execute because the session does not support password substitution, the Program returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
AP_PW_SUB_NOT_SUPP_ON_SESS

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_LU_LU_PASSWORD

DEFINE_LU_LU_PASSWORD provides a password that is used for session-level verification between a local LU and a partner LU.

VCB Structure

```
typedef struct define_lu_lu_password
{
   unsigned short  opcode;           /* verb operation code        */
   unsigned char   reserv2;          /* reserved                   */
   unsigned char   format;           /* format                     */
   unsigned short  primary_rc;       /* primary return code        */
   unsigned long   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
                                     /* LULU verification protocol */
   unsigned char   description[RD_LEN];
                                     /* resource description       */
   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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Local LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_alias** and the **lu_name** are set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

fqplu_name
Fully qualified partner LU name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

verification_protocol
LU-LU verification protocol for use with this partner LU:

AP_BASIC_PROTOCOL
Only the basic protocol will be used with this partner LU.

AP_ENHANCED_PROTOCOL
Only the enhanced protocol will be used with this partner LU.

AP_EITHER_PROTOCOL

Either the basic or the enhanced protocol can be used with this partner LU, subject to the following details:

- The default setting of this field is AP_EITHER_PROTOCOL.
- The value AP_EITHER_PROTOCOL is provided to ease migration to the use of the enhanced protocol. The local LU accepts the basic protocol until the partner LU once agrees to run the enhanced protocol. From then on, the basic protocol is not accepted unless a subsequent DEFINE_LU_LU_PASSWORD is issued to allow it.

description

Resource description.

password

Password. This is an 8-byte hexadecimal string. Note that the least significant bit of each byte in the password is not used in session-level verification.

Returned Parameters

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

primary_rc

AP_OK

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

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_PLU_NAME

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc

AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc

AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR

DEFINE_USERID_PASSWORD

DEFINE_USERID_PASSWORD defines a password associated with a user ID.

VCB Structure

```
define_userid_password
{
   unsigned short  opcode;              /* verb operation code       */
   unsigned char   reserv2;             /* reserved                  */
   unsigned char   format;              /* format                    */
   unsigned short  primary_rc;          /* primary return code       */
   unsigned long   secondary_rc;        /* secondary return code     */
   unsigned short  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[RD_LEN];
                                        /* resource description      */
   unsigned short  profile_count;       /* number of profiles        */
   unsigned short  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

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

define_type
Specifies the type of user password being defined:

AP_ADD_USER
Specifies a new user, or change of password for an existing user.

AP_ADD_PROFILES
Specifies an addition to the profiles for an existing user.

user_id
User identifier. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

password_chars.description
Resource description. This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

password_chars.profile_count
Number of profiles.

password_chars.password
User's password. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

password_chars.profiles
Profiles associated with user. Each of these is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_NO_PROFILES

AP_UNKNOWN_USER
AP_INVALID_UPDATE_TYPE
AP_TOO_MANY_PROFILES
AP_INVALID_USERID
AP_INVALID_PROFILE
AP_INVALID_PASSWORD

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_LU_LU_PASSWORD

DELETE_LU_LU_PASSWORD deletes an LU-LU password.

VCB Structure

```
typedef struct delete_lu_lu_password
{
   unsigned short  opcode;          /* verb operation code       */
   unsigned char   reserv2;         /* reserved                  */
   unsigned char   format;          /* format                    */
   unsigned short  primary_rc;      /* primary return code       */
   unsigned long   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

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Local LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_alias** and the **lu_name** are set to all zeros, the verb is forwarded to the LU associated with the control point (the default LU).

fqplu_name
Fully qualified partner LU name. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PLU_NAME

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_USERID_PASSWORD

DELETE_USERID_PASSWORD deletes a password associated with a user ID.

VCB Structure

```
typedef struct delete_userid_password
{
   unsigned short  opcode;              /* verb operation code       */
   unsigned char   reserv2;             /* reserved                  */
   unsigned char   format;              /* format                    */
   unsigned short  primary_rc;          /* primary return code       */
   unsigned long   secondary_rc;        /* secondary return code     */
   unsigned short  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[RD_LEN]; /* resource description      */
   unsigned short  profile_count;       /* number of profiles        */
   unsigned short  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

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

delete_type
Specifies the type of delete:

AP_REMOVE_USER
Deletes the user password, and all associated profiles.

AP_REMOVE_PROFILES
Deletes the specified profiles.

user_id
User identifier. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

password_chars.description
This field is ignored when processing this verb.

password_chars.profile_count
Number of profiles.

password_chars.password
This field is ignored when processing this verb.

password_chars.profiles
Profiles associated with user. Each of these is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_NO_PROFILES

AP_UNKNOWN_USER
AP_INVALID_UPDATE_TYPE

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SIGN_OFF

SIGN_OFF instructs an LU to remove entries from signed on lists. Currently, only entries from the signed-on list are removed. The verb can specify that all entries are removed, or that only those in the appended sign_off_data structures.

VCB Structure

```
typedef struct query_sign_off
{
   unsigned short  opcode;           /* verb operation code          */
   unsigned char   reserv2;          /* reserved                     */
   unsigned char   format;           /* format                       */
   unsigned short  primary_rc;       /* primary return code          */
   unsigned long   secondary_rc;     /* secondary return code        */
   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   list;             /* signed on to/from list       */
   unsigned char   all_in_list;      /* sign off all entries in list */
   unsigned char   immediate;        /* remove entries immediately   */
   unsigned char   num_entries;      /* number of entries            */
} QUERY_SIGN_OFF;

typedef struct sign_off_data
{
   unsigned char   user_id[10];      /* user ID                      */
   unsigned char   all_profiles;    /* all profiles for this user    */
   unsigned char   profile[10];     /* specific profile             */
} SIGN_OFF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_SIGN_OFF

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the index.

lu_alias
Locally defined LU alias. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant and must be set. If both the **lu_name** and the **lu_alias** fields are set to all zeros, the LU associated with the control point (the default LU) is used.

plu_alias
Partner LU alias. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field will be used for determining the index.

fqplu_name
17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an

EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

list Signed–on list type. This must be set to AP_SIGNED_ON_TO_LIST.

AP_SIGNED_ON_TO_LIST
The list of users who are signed on to the remote LU from the local LU. Note, the remote LU is not informed when entries are removed from this list. This is the only value currently supported.

all_in_list
If set to AP_YES, all users in the list specified by **list** are signed off.

immediate
If set to AP_YES, users are removed immediately. If set to AP_NO, users are removed once the remote LU has confirmed that the sign-off completed successfully. This field is reserved if **list** is AP_SIGNED_ON_TO_LIST.

num_entries
Number of entries actually returned.

If **all_in_list** is AP_NO, a list of users must be appended to the SIGN_OFF VCB, as a series of SIGN_OFF_DATA structures. The parameters in the SIGN_OFF_DATA structure are as follows:

sign_off_data.user_id
The user ID.

sign_off_data.all_profiles
Total number of entries that could have been returned. This can be higher than **num_entries**.

sign_off_data.profile
This is a 10-byte alphanumeric EBCDIC string. Note, the Program currently supports only the blank profile (10 eBCDIC spaces). This field is ignored if **list_options** is set to AP_FIRST_IN_LIST.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_ALIAS

AP_INVALID_LU_NAME
AP_INVALID_PLU_NAME
AP_INVALID_USERID
AP_INVALID_PROFILE
AP_INVALID_LIST
AP_INVALID_LIST_OPTION

Any SIGN_OFF_DATA **user_id/profile** combinations that are not successfully processed by the Program, are returned to the application appended to the VCB, and the returned value of **num_entries** is the number of SIGN_OFF_DATA entries (which could not be processed) returned by the Program.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_ALIAS

AP_INVALID_LU_NAME
AP_INVALID_LU_NAME
AP_INVALID_LIST

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node stopped, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 11. APING and CPI-C Verbs

This chapter describes verbs used to ping another node and verbs used to define, delete, and query CPI-C side information.

APING

APING allows a management application to ping a remote LU in the network. A verification data string (of specified length) can be appended to the end of the VCB and returned when the **partner_ver_len** field is set to a value greater than zero.

Personal Communications or Communications Server APING is implemented as an internal service transaction program, which uses the Personal Communications or Communications Server APPC API (described in the *Personal Communications Client/Server Communications Programming*).

VCB Structure

```
typedef struct aping
{
   unsigned short  opcode;           /* verb operation code          */
   unsigned char   reserv2;          /* reserved                     */
   unsigned char   format;           /* format                       */
   unsigned short  primary_rc;       /* primary return code          */
   unsigned long   secondary_rc;     /* secondary return code        */
   unsigned char   lu_name[8];       /* local LU name                */
   unsigned char   lu_alias[8];      /* local LU alias               */
   unsigned long   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                      */
   unsigned short  dlen;             /* length of data to send       */
   unsigned short  consec;           /* number of consecutive sends  */
   unsigned char   fqplu_name[17];   /* fully qualified partner      */
                                     /* LU name                      */
   unsigned char   echo;             /* data echo flag               */
   unsigned short  iterations;       /* number of iterations         */
   unsigned long   alloc_time;       /* time taken for ALLOCATE      */
   unsigned long   min_time;         /* min send/receive time        */
   unsigned long   avg_time;         /* average send/receive time    */
   unsigned long   max_time;         /* max send/receive time        */
   unsigned short  partner_ver_len;  /* size of string to receive    */
} APING;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_APING

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

lu_name
LU name of the local LU from which the APING verb is sent. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, the **lu_alias** field will be used for determining the local LU.

lu_alias
Alias for the local LU from which the APING verb is sent. This is an 8-byte string in a locally displayable character set. This field is only significant if the **lu_name** field is set to all zeros, in which case all 8 bytes are significant

and must be set. If both the **lu_name** and the **lu_alias** are set to binary zeros then the default (control point) LU is used.

plu_alias
Alias by which the partner LU is known to the local transaction program. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. This name must match the name of a partner LU established during configuration. If this parameter is set to binary zeros, the **fqplu_name** parameter is used instead.

mode_name
Name of the mode to be used. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

tp_name
Name of the invoked transaction program. This is a 64-byte string. The Node Operator Facility does not check the character set of this string. The value of **tp_name** must match that configured on the remote LU. The string is usually set to APINGD in EBCDIC padded to the right with EBCDIC spaces.

security
Specifies the information the partner LU requires in order to validate access to the invoked transaction program:

AP_NONE
AP_PGM
AP_SAME
AP_PGM_STRONG

pwd Password associated with **user_id**. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. Only needed if **security** is set to AP_PGM or AP_PGM_STRONG.

user_id
User ID required to access the partner transaction program. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. Needed if **security** is set to AP_PGM, AP_PGM_STRONG or AP_SAME.

dlen Length of data to be sent by APING transaction program. APING sends a string of zeros, of length **dlen**.

consec Number of consecutive sends performed during each iteration. APING issues this number of MC_SEND_DATA verbs, each consisting of **dlen** bytes of data. If the **echo** parameter is set to AP_YES, APING marks the last MC_SEND_DATA as AP_SEND_DATA_P_TO_R_FLUSH (Prepare to Receive Flush) and awaits a response containing data from the partner APINGD transaction program (by issuing a MC_RECEIVE_AND_WAIT). If the **echo** parameter is set to AP_NO, APING flushes the data and awaits a confirm (by marking the last MC_SEND_DATA as AP_SEND_DATA_CONFIRM). In either case, the sequence described here corresponds to an SNA chain.

fqplu_name
This is a 17-byte fully qualified network name for the partner LU. This name is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot, and is right-padded with EBCDIC spaces. (Each name

can have a maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

echo Specifies whether the APING transaction program expects a response when it has completed sending the required amount of data:

AP_YES
AP_NO

iterations
Number of iterations of consecutive sequences (defined by the **consec** parameter) issued by APING. In SNA terms, this parameter defines the number of chains that will be sent.

partner_ver_len
Maximum length of the partner transaction program verification data string that can be received by the management application.

Returned Parameters

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

primary_rc
AP_OK

sense_data
This will be zero if the verb has returned successfully.

alloc_time
Time required (in milliseconds) for the MC_ALLOCATE to the remote transaction program to complete.

min_time
Minimum time (in milliseconds) required for a data-sending iteration. This parameter includes the time required for the partner to respond (either by sending data or issuing a confirm, depending on the setting of the **echo** parameter).

avg_time
Average time (in milliseconds) required for a data-sending iteration. This parameter includes the time required for the partner to respond (either by sending data or issuing a confirm, depending on the setting of the **echo** parameter).

max_time
Maximum time (in milliseconds) required for a data-sending iteration. This parameter includes the time required for the partner to respond (either by sending data or issuing a confirm, depending on the setting of the **echo** parameter).

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

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_LU_NAME

AP_INVALID_LU_ALIAS

APING uses the MC_ALLOCATE, MC_SEND_DATA, MC_RECEIVE_AND_WAIT, MC_CONFIRM, and MC_DEALLOCATE verbs provided by the Personal Communications or Communications Server APPC API. The parameters returned by these verbs in the case of unsuccessful execution are documented in the *Personal Communications Client/Server Communications Programming*.

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

CPI-C Verbs

This section describes the verbs used to define, delete, and query CPI-C side information.

DEFINE_CPIC_SIDE_INFO

This verb adds or replaces a side information entry in memory. 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 in memory with the same symbolic destination name as the one supplied with this verb, it is overwritten with the data supplied to this call. Refer to *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications or Communications Server.

VCB Structure

```
typedef struct define_cpic_side_info
{
   unsigned short  opcode;            /* verb operation code          */
   unsigned char   reserv2;           /* reserved                     */
   unsigned char   format;            /* format                       */
   unsigned short  primary_rc;        /* primary return code          */
   unsigned long   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;  /* defined data                 */
} DEFINE_CPIC_SIDE_INFO;

typedef struct cpic_side_info_def_data
{
   unsigned char   description[RD_LEN];
                                      /* resource description         */
   CPIC_SIDE_INFO  side_info;         /* CPIC side info               */
   unsigned char   user_data[32];     /* User defined data            */
} 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                     */
   unsigned long   tp_name_type;      /* TP name type                 */
   unsigned char   tp_name[64];       /* TP name                      */
   unsigned char   mode_name[8];      /* Mode name                    */
   unsigned long   conversation_security_type;
                                      /* Conversation security type   */
   unsigned char   security_user_id[CPIC_SECURITY_INFO_LEN];
                                      /* User ID                      */
   unsigned char   security_password[CPIC_SECURITY_INFO_LEN];
                                      /* Password                     */
} CPIC_SIDE_INFO;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DEFINE_CPIC_SIDE_INFO

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

sym_dest_name
Symbolic destination name that identifies the side information entry. This is up to 8 bytes long, padded with spaces, in the locally displayable character set. The allowed characters are the uppercase letters (A to Z) and the digits 0–9.

def_data.description
Resource description (returned on QUERY_CPIC_SIDE_INFO). This is a 16-byte string in a locally displayable character set. All 16 bytes are significant.

def_data.side_info.partner_lu_name
Fully qualified name of the partner LU. This name is 17 bytes long and is right-padded with spaces, in the locally displayable character set. It is composed of two character strings concatenated by a dot. (Each name can have a maximum length of 8 bytes with no embedded spaces.)

def_data.side_info.tp_name_type
Transaction program name type. This field is set to one of the following values:

XC_APPLICATION_TP
Specifies that the transaction program name supplied is not a service transaction program. All characters specified in the transaction program name must be valid characters in the locally displayable character set.

XC_SNA_SERVICE_TP
Specifies that the transaction program name supplied is that of a service transaction program. All characters, except the first, specified in the transaction program must be valid characters in the locally displayable character set. The first character must be a hexadecimal digit in the range X'01' to X'3F', excluding X'0E' and X'0F'.

def_data.side_info.tp_name
Transaction program name, a 64-byte character string in the locally displayable character set, right-padded with spaces.

def_data.side_info.mode_name
Mode name, an 8-byte character string in the locally displayable character set, padded to the right with spaces.

def_data.side_info.conversation_security_type
Conversation security type. This field is set to one of the following values:

XC_SECURITY_NONE
XC_SECURITY_SAME
XC_SECURITY_PROGRAM
XC_SECURITY_PROGRAM_STRONG.

def_data.side_info.security_user_id
User ID. Personal Communications or Communications Server will use this field for enforcing conversation-level security.

def_data.side_info.security_password
Password. Personal Communications or Communications Server will use this field for enforcing conversation-level security.

def_data.user_data
User data. This data is returned on QUERY_CPIC_SIDE_INFO but not used or interpreted by Personal Communications or Communications Server.

Returned Parameters

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

primary_rc
AP_OK

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_SYM_DEST_NAME

AP_INVALID_LENGTH

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

DELETE_CPIC_SIDE_INFO

This verb deletes a CPI-C side information entry. Refer to *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications or Communications Server.

VCB Structure

```
typedef struct delete_cpic_side_info
{
   unsigned short  opcode;            /* verb operation code          */
   unsigned char   reserv2;           /* reserved                     */
   unsigned char   format;            /* format                       */
   unsigned short  primary_rc;        /* primary return code          */
   unsigned long   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

format
: Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

sym_dest_name
: Symbolic destination name that identifies the side information entry. This is up to 8 bytes long, padded with spaces, in the locally displayable character set. The allowed characters are the uppercase letters (A to Z) and the digits 0–9.

Returned Parameters

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

primary_rc
: AP_OK

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

primary_rc
: AP_STATE_CHECK

secondary_rc
: AP_INVALID_SYM_DEST_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
: AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
: AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_CPIC_SIDE_INFO

This verb returns the side information entry for a given symbolic destination name. The information is returned as a list. To obtain a specific side information entry, or a specific chunk of entries, the **sym_dest_name** field should be set. Otherwise this field should be set to all zeros.

VCB Structure

```
typedef struct query_cpic_side_info
{
   unsigned short  opcode;               /* verb operation code       */
   unsigned char   reserv2;              /* reserved                  */
   unsigned char   format;               /* format                    */
   unsigned short  primary_rc;           /* primary return code       */
   unsigned long   secondary_rc;         /* secondary return code     */
   unsigned char   *buf_ptr;             /* pointer to buffer         */
   unsigned long   buf_size;             /* buffer size               */
   unsigned long   total_buf_size;       /* total buffer size required */
   unsigned short  num_entries;          /* number of entries         */
   unsigned short  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
{
   unsigned short  overlay_size;         /* size of this 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
{
   unsigned char   partner_lu_name[17];
                                         /* Fully qualified partner    */
                                         /* LU name                   */
   unsigned char   reserved[3];          /* Reserved                  */
   unsigned long   tp_name_type;         /* TP name type              */
   unsigned char   tp_name[64];          /* TP name                   */
   unsigned char   mode_name[8];         /* Mode name                 */
   unsigned long   conversation_security_type;
                                         /* Conversation security type */
   unsigned char   security_user_id[CPIC_SECURITY_INFO_LEN];
                                         /* User ID                   */
   unsigned char   security_password[CPIC_SECURITY_INFO_LEN];
                                         /* Password                  */
} CPIC_SIDE_INFO;
```

```
typedef struct cpic_side_info_def_data
{
   unsigned char   description[RD_LEN];
                                         /* resource description      */
   CPIC_SIDE_INFO  side_info;            /* CPIC side info            */
   unsigned char   user_data[32];        /* User defined data         */
} CPIC_SIDE_INFO_DEF_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_QUERY_CPIC_SIDE_INFO

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

buf_ptr
Pointer to a buffer into which list information can be written.

buf_size
Size of buffer supplied. The data returned will not exceed this size.

num_entries
Maximum number of entries to return. The number of entries will not exceed this value. A value of zero means no limit.

list_options
This indicates what should be returned in the list information. The **sym_dest_name** specified (see below) represents an index value that is used to specify the starting point of the actual information to be returned:

AP_FIRST_IN_LIST
The index value is ignored and the returned list starts from the first entry in the list.

AP_LIST_FROM_NEXT
The returned list starts from the next entry in the list after the one specified by the supplied index value.

AP_LIST_INCLUSIVE
The returned list starts from the entry specified by the index value.

sym_dest_name
Symbolic destination name that identifies the side information entry. This is up to 8 bytes long, padded with spaces, in the locally displayable character set. The allowed characters are the uppercase letters (A to Z) and the digits 0-9.

Returned Parameters

If the verb executes successfully, the Program 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
Number of entries actually returned.

total_num_entries
Total number of entries that could have been returned. This may be higher than **num_entries**.

cpic_side_info_data.overlay_size
The number of bytes in this entry, and hence the offset to the next entry returned (if any).

cpic_side_info_data.sym_dest_name
Symbolic destination name for the returned side information entry.

cpic_side_info_data.def_data
Defined CPI-C side information as supplied on DEFINE_CPIC_SIDE_INFO verb.

Note: CPIC calls may change the side information returned on this verb after the DEFINE_CPIC_SIDE_INFO has been processed by Personal Communications or Communications Server.

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

primary_rc
AP_STATE_CHECK

secondary_rc
AP_INVALID_SYM_DEST_NAME

If the verb does not execute because the node has not yet been started, the Program returns the following parameter:

primary_rc
AP_NODE_NOT_STARTED

If the verb does not execute because the node is stopping, the Program returns the following parameter:

primary_rc
AP_NODE_STOPPING

If the verb does not execute because of a system error, the Program returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 12. Attach Manager Verbs

The Personal Communications or Communications Server Attach Manager is used to manage the launching of APPC or CPI-C programs. A description of the Attach Manager function is provided in *Personal Communications Client/Server Communications Programming*.

Personal Communications or Communications Server Node Operator Facility supports three verbs to control the Attach Manager. These verbs are available to any application program that uses Personal Communications or Communications Server Node Operator Facility.

DISABLE_ATTACH_MANAGER

The Personal Communications or Communications Server Attach Manager is enabled by default when the node is started. The user can issue this verb to disable all dynamic loading, This verb resets a global flag that the Attach Manager checks before launching a transaction program.

VCB Structure

```
typedef struct disable_am
{
   unsigned short   opcode;              /* Verb operation code      */
   unsigned char    reserv2;             /* reserved                 */
   unsigned char    format;              /* format                   */
   unsigned short   primary_rc;          /* Primary return code      */
   unsigned long    secondary_rc;        /* Secondary return code    */
} DISABLE_AM;
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_DISABLE_ATTACH_MGR

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

If the verb executes successfully, the Attach Manager returns the following parameter:

primary_rc
AP_OK

If the verb does not execute because the node has not yet been started, the Attach Manager returns the following parameter:

primary_rc
AP_NODE_NOT STARTED

If the verb does not execute because of a system error, the Attach Manager returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

ENABLE_ATTACH_MANAGER

If the Attach Manager has been disabled, it can be re-enabled by issuing the Personal Communications or Communications Server Node Operator Facility verb, ENABLE_AM. This sets a global flag that the Attach Manager checks before launching a Transaction Program.

VCB Structure

```
typedef struct enable_am
{
   unsigned short  opcode;          /* Verb operation code       */
   unsigned char   reserv2;         /* reserved                  */
   unsigned char   format;          /* format                    */
   unsigned short  primary_rc;      /* Primary return code       */
   unsigned long   secondary_rc;    /* Secondary return code     */
} ENABLE_AM
```

Supplied Parameters

The application supplies the following parameters:

opcode
AP_ENABLE_ATTACH_MGR

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

If the verb executes successfully, the Attach Manager returns the following parameter:

primary_rc
AP_OK

If the verb does not execute because the node has not yet been started, the Attach Manager returns the following parameter:

primary_rc
AP_NODE_NOT STARTED

If the verb does not execute because of a system error, the Attach Manager returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_ATTACH_MANAGER

The QUERY_ATTACH_MANAGER verb can be used to discover the status of the Attach Manager component, which can be started and stopped using the ENABLE_ATTACH_MANAGER and DISABLE_ATTACH_MANAGER commands.

VCB Structure

```
typedef struct query_am
{
        unsigned short   opcode;          /* Verb operation code          */
        unsigned char    reserv2;         /* reserved                     */
        unsigned char    format;          /* format                       */
        unsigned short   primary_rc;      /* primary return code          */
        unsigned long    secondary_rc;    /* secondary return code        */
        unsigned short   active;          /* status of the Attach Manager */
} QUERY_AM;
```

Supplied Parameters

opcode
AP_QUERY_ATTACH_MGR

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

Returned Parameters

If the verb executes successfully, the following parameters are returned:

primary_rc
AP_OK

active This field reports the status of the Attach Manager component:

AP_YES
The Attach Manager is active.

AP_NO
The Attach Manager is not active.

If the verb does not execute because of a parameter error, the following parameter is returned:

primary_rc
AP_PARAMETER_CHECK

If the verb does not execute because the node has not yet been started, the Attach Manager returns the following parameter:

primary_rc
AP_NODE_NOT STARTED

If the verb does not execute because of a system error, the Attach Manager returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Part 2. Personal Communications and Communications Server Management Services API

Chapter 13. Introduction to Management Services API

This chapter describes the management services API provided by Personal Communications or Communications Server.

Management Services Verbs

Personal Communications or Communications Server supports the following management services (MS) verbs, providing an application program with a method for reporting potential problems to management services focal points available in an SNA network.

- ALERT_INDICATION
- FP_INDICATION
- MDS_MU_RECEIVED
- NMVT_RECEIVED
- SEND_MDS_MU
- TRANSFER_MS_DATA

Entry Points

Personal Communications or Communications Server provides a library file that handles management services verbs.

Management services verbs have a straightforward language interface. Your program fills in fields in a block of memory called a *verb control block* Then your program calls the entry point and passes a pointer to the verb control block. When its operation is complete, management services (MS) API returns, having used and then modified the fields in the verb control block. Your program can then read the returned parameters from the verb control block. Following is a list of entry points for management services verbs:

- WinMS()
- WinMSCleanup()
- WinMSGetIndication()
- WinMSRegisterApplication()
- WinMSStartup()
- WinMSUnregisterApplication()

WINMS is the only API supported on the Windows 2000, Windows 95, and NT clients; see "WinMS()" on page 602 for more information.

See Chapter 14, "Management Services Entry Points", on page 601 for detailed descriptions of the entry points.

Verb Control Blocks (VCB)

Programming Note: The base operating system optimizes performance by executing some subsystems in the calling application's address space. This means that incorrect use of local descriptor table (LDT) selectors by application programs that have not been fully or correctly debugged can cause improper operation, or perhaps system failures.Accordingly, application programs should not perform pointer arithmetic operations that involve changing the LDT selector field of a pointer.

The segment used for the verb control block (VCB) must be a read/write data segment. Your program can either declare the VCB as a variable in your program, allocate it or suballocate it from a larger segment. It must be sufficiently large to contain all the fields for the verb your program is issuing.

An application program should not change any part of the verb control block after it has been issued until the verb completes. When management services finishes the execution of a verb, it copies a complete, modified VCB back onto the original block. Therefore, if your program declares a verb control block as a variable, consider declaring it in static storage rather than on the stack of an internal procedure.

Fill all reserved and unused fields in each VCB with zeros (X'00'). In fact, it might be more time-efficient to set the entire verb control block to zeros before your program assigns the values to the parameters. Setting reserved fields to zeros is particularly important.

Note: If the VCB is not read/write, or if it is not at least 10 bytes (that is, large enough to hold the management services primary and secondary return codes), management services cannot access it, and the base operating system abnormally ends the process. This termination is recognized as a *general protection fault*, processor exception trap D.

Management services returns the INVALID_VERB_SEGMENT primary return code when the VCB is too short or the incorrect type of segment is used.

Writing Management Services (MS) Programs

Personal Communications or Communications Server provides a dynamic link library (DLL) file, that handles Management Services verbs.

The DLL is reentrant; multiple application processes and threads can call the DLL concurrently.

Management Services verbs have a straightforward language interface. Your program fills in fields in a block of memory called a *verb control block* (VCB). Then it calls the Management Services DLL and passes a pointer to the verb control block. When its operation is complete, Management Services returns, having used and then modified the fields in the VCB. Your program can then read the returned parameters from the verb control block.

Table 3 shows source module usage of supplied header files and libraries needed to compile and link Management Services programs. Some of the header files may include other required header files.

Table 3. Header Files and Libraries for Management Services

Operating System*	**Header File**	**Library**	**DLL Name**
WIN32	WINMS.H	WINMS32.LIB	WINMS32.DLL
WIN3.1	WINCSV.H	WINCSV.LIB	WINCSV.DLL
OS/2®	ACSSVCC.H	ACSSVC.LIB	ACSSVC.DLL

* WIN32 = Windows 95, Windows 98, Windows NT, Windows Me, and Windows 2000

*WIN3.1 = Windows 3.1 (available only on Communications Server)

SNA API Client Support

Included with Communications Server are a set of clients for the Windows 2000, Windows 95, Windows NT, and OS/2 operating systems. These clients are referred to as SNA API clients in this book and only support a subset of the full management services verbs. Specifically, **WINMS** is the only API supported on the Windows 2000, Windows 95, and NT clients, see "WinMS()" on page 602 for more information.

The following is a list of the management services verbs supported:

- TRANSFER_MS_DATA
- SEND_MDS_MU

Chapter 14. Management Services Entry Points

This chapter describes the entry points for management services verbs.

WinMS()

This is the only entry point supported for Windows 95, Windows 98, Windows NT, Windows Me, and Windows 2000.

This provides a synchronous entry point for issuing the following management services API verbs:

- SEND_MDS_MU
- TRANSFER_MS_DATA

Syntax

```
void WINAPI WinMS(long vcb, unsigned short vcb_size);
```

Parameters

vcb Pointer to verb control block

vcb_size
Number of bytes in the verb control block

Returns

No return value. The **primary_rc** and **secondary_rc** fields in the verb control block indicate any error.

Remarks

This is the main synchronous entry point for the management services API. This call blocks until the verb completes.

WinMSCleanup()

This function terminates and unregisters an application from the management services API.

Syntax

```
BOOL WINAPI WinMSCleanup(void);
```

Returns

The return value specifies whether the unregistration was successful. If the value is not zero, the application was successfully unregistered. The application was not unregistered if a value of zero is returned.

Remarks

Use **WinMSCleanup()** to indicate unregistration of a management services application from the management services API.

WinMSCleanup unblocks any thread waiting in **WinMSGetIndication**. These return with WMSNOTREG (the application is not registered to receive indication). **WinMSCleanup** unregisters the application for all indications. **WinMSCleanup** returns any outstanding verb (synchronous or asynchronous) with the error AP_CANCELLED. However, the verb completes inside the node.

It is not a requirement to use **WinMSStartup** and **WinMSCleanup**. However, an application must be consistent in its use of these calls. You should use both of them or never use either of them.

Note: See also **WinMSStartup()**.

WinMSGetIndication()

This allows the application to receive unsolicited indications.

Syntax

```
int WINAPI WinMSGetIndication(long buffer,
                              unsigned short *buffer_size,
                              unsigned long  timeout);
```

Parameters

buffer Pointer to a buffer into which to receive the indication.

buffer_size
Size of buffer. Returned: the size of the indication.

timeout
Time to wait for indication in milliseconds.

Returns

The function returns a value indicating whether an indication was received.

0 Indication returned.

WMSTIMEOUT
Timeout waiting for indication.

WMSSYSNOTREADY
The underlying network subsystem is not ready for network communication.

WMSNOTREG
The application is not registered to receive indications.

WMSBADSIZE
The buffer is too small to receive the indication. Reissue the **WinMSGetIndication** call with a large enough buffer. The size of the indication is returned in the **buffer_size** parameter.

WMSBADPOINTER
Either the buffer or **buffer_size** parameter is not valid.

WMSSYSERROR
An unexpected system error has occurred.

Remarks

This is a blocking call, it returns in one of the following circumstances:

- An indication is returned
- The timeout expires
- The application issues a **WinMSCleanup** call
- The product is stopped
- A system error occurs

Note: See also **WinMSRegisterApplication** and **WinMSUnregisterApplication**.

WinMSRegisterApplication()

This function registers the application as an NMVT-level application, an MDS-level application, or an alert handler. Such registrations determine which unsolicited indications the application receives.

- An NMVT-level application receives NMVT_RECEIVED indications.
- An MDS-level application receives MDS_MU_RECEIVED indications and also FP_NOTIFICATION indications when focal-point status changes.
- An alert handler receives ALERT_INDICATION indications.

Note: It is also possible to register to receive NMVTs with conversion to MDS MUs.

Applications that do not process these indications should not call **WinMSRegisterApplication**.

Syntax

```
BOOL WINAPI WinMSRegisterApplication(unsigned short reg_type,
                                     unsigned char  *ms_appl_name,
                                     unsigned short vector_key,
                                     unsigned char  mds_conv_reqd,
                                     unsigned char  *ms_category,
                                     unsigned short max_rcv_size,
                                     unsigned char  alert_dest,
                                     unsigned short *primary_rc,
                                     unsigned long  *secondary_rc);
```

Parameters

reg_type
Registration type

WMSNMVTAPP	NMVT-level application (or MDS-level application registering to receive NMVTs)
WMSMDSAPP	MDS-level application
WMSALERTHANDLER	Alert handler

ms_appl_name
Management services application name. Valid names can be either an 8-byte alphanumeric type-1134 EBCDIC string, padded with trailing space (X'40') characters if necessary, or one of the management services discipline-specific application programs specified in *SNA Management Services Reference* padded with trailing space (X'40') characters.

This name is used when **reg_type** is WMSNMVTAPP or WMSMDSAPP. The name is not applicable when **reg_type** is WMSALERTHANDLER.

vector_key
Management services major vector keys accepted by the application Permitted values are:

X'YYYY'	specific major vector key
AP_SPCF_KEYS	major vector keys X'8061' through X'8064'
AP_ALL_KEYS	all major vector keys

This key is used when **reg_type** is WMSNMVTAPP. The key is not applicable when **reg_type** is WMSMDSAPP or WMSALERTHANDLER.

mds_conv_reqd
Specifies whether the registering application is MDS-level and requires NMVTs sent to it to be converted to MDS MUs

`(AP_YES or AP_NO)`

This parameter is used when **reg_type** is `WMSNMVTAPP`. The parameter is not applicable when **reg_type** is `WMSMDSAPP` or `WMSALERTHANDLER`.

ms_category
Specifies a management services category when the application desires information pertaining to the focal point for that category. The management services category can be either one of the category codes specified in the management services discipline-specific application programs table provided in *SNA Management Services Reference* padded with trailing space (X'40') characters or a user-defined category. User-defined category names should be an 8-byte alphanumeric type-1134 EBCDIC string, padded with trailing space (X'40') characters if necessary.

This parameter is used when **reg_type** is `WMSMDSAPP`. The parameter is not applicable when **reg_type** is `WMSNMVTAPP` or `WMSALERTHANDLER`.

max_rcv_size
Maximum number of bytes the application is capable of receiving in one chunk. MDS MUs bigger that this size will be segmented, and each segment delivered in a separate MDS_MU_RECEIVED indication.

This parameter is used when **reg_type** is `WMSMDSAPP`. The parameter is not applicable when **reg_type** is `WMSNMVTAPP` or `WMSALERTHANDLER`.

alert_dest
Specifies whether the application wishes to be the only destination of all alerts. If this is set to AP_YES then all alerts will be routed to the application, and will not be routed anywhere else. If set to AP_NO, alerts will be routed to the application and over the SNA network in the usual way.

This parameter is used when **reg_type** is `WMSALERTHANDLER`. The parameter is not applicable when **reg_type** is `WMSNMVTAPP` or `WMSMDSAPP`.

primary_rc
Returned: primary return code

secondary_rc
Returned: secondary return code

Returns

The function returns a value indicating whether the registration was successful. If the value is not zero, the registration was successful. If the value is zero, the registration was not successful.

Remarks

Applications can make multiple calls to register more than one class of indications.

Applications that call **WinMSRegisterApplication** must call **WinMSGetIndication** to receive indications that are queued for them.

Note: See also **WinMSUnregisterApplication** and **WinMSGetIndication**.

WinMSStartup()

This function allows an application to specify the version of management services API required and to retrieve the version of the API supported by the product. This function can be called by an application before issuing any further management services API calls to register itself.

Syntax

```
int WINAPI WinMSStartup(WORD wVersionRequired,
                        LPWMSDATA msdata);
```

Parameters

wVersionRequired
Specifies the version of management services API support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.

msdata
Returns the version of management services API and a description of management services implementation.

Returns

The return value specifies whether the application was registered successfully and whether the management services API implementation can support the specified version number. If the value is zero, it was registered successfully and the specified version can be supported. Otherwise, the return value is one of the following values:

WMSSYSERROR
The underlying network subsystem is not ready for network communication.

WMSVERNOTSUPPORTED
The version of management services API support requested is not provided by this particular management services API implementation.

WMSBADPOINTER
Incorrect msdata parameter.

Remarks

WinMSStartup is intended to help with compatibility with future versions of the API. The current version supported is 1.0.

It is not a requirement to use **WinMSStartup** and **WinMSCleanup**. However, an application must be consistent in its use of these calls. You should use both of them or never use either of them.

Note: See also **WinMSCleanup()**.

WinMSUnregisterApplication()

This function unregisters the application, reversing the effect of an earlier **WinMSRegisterApplication** call, and stopping further indications from being queued for the application.

Syntax

```
BOOL WINAPI WinMSUnregisterApplication(unsigned short reg_type,
                                       unsigned char  *ms_appl_name,
                                       unsigned short *primary_rc,
                                       unsigned long  *secondary_rc);
```

Parameters

reg_type
Registration type. It can have one of the following values:

WMSNMVTAPP
NMVT-level application

WMSMDSAPP
MDS-level application

WMSALERTHANDLER
Alert handler

ms_appl_name
MS application name. Valid names can be either an 8-byte alphanumeric type-1134 EBCDIC string, padded with trailing space (X'40') characters if necessary, or one of the management services discipline-specific application programs specified in *SNA Management Services Reference* padded with trailing space (X'40') characters.

This parameter is used when **reg_type** is WMSNMVTAPP or WMSMDSAPP. The parameter is not applicable when **reg_type** is WMSALERTHANDLER.

primary_rc
Returned: primary return code

secondary_rc
Returned: secondary return code

Returns

The function returns a value indicating whether the unregistration was successful. If the value is not zero, the unregistration was successful. If the value is zero, the unregistration was not successful.

Remarks

Each call to **WinMSUnregisterApplication** terminates a registration made by an earlier call to **WinMSRegisterApplication**. An application that has made multiple calls to **WinMSRegisterApplication** needs to make multiple calls to **WinMSUnregisterApplication** in order to terminate all its registrations.

WinMSUnregisterApplication and **WinMSCleanup** differ as follows:

- **WinMSUnregisterApplication** terminates an earlier registration to receive indications, but does not prevent the application from making other management services API calls (for example, WinMS).
- **WinMSCleanup** terminates use of the management services API.

Indications might already be queued for an application when the application calls **WinMSUnregisterApplication**. Any such indications remain queued, and the application should call **WinMSGetIndication** to receive and process them. Once they have been unregistered, no new indications will be queued for the application.

Note: See also **WinMSRegisterApplication** and **WinMSGetIndication**.

Chapter 15. Management Services Verbs

The management services API verbs provided by Personal Communications or Communications Server enable an application to send alerts and MDS MU’s, and to receive indications when the node receives MDS or NMVT data or issues an alert.

ALERT_INDICATION

This verb indication is used by management services to send alert major vectors to a registered alert handler or registered held alert handler that will process them.

VCB Structure

```
typedef struct ms_alert_indication
{
   unsigned short   opcode;              /* AP_AlERT_INDICATION         */
   unsigned char    reserv2;             /* reserved                    */
   unsigned char    format;              /* format                      */
   unsigned short   primary_rc;          /* Primary return code         */
   unsigned long    secondary_rc;        /* Secondary return code       */
   unsigned short   alert_length;        /* Length of alert             */
   unsigned char    reserv3[6];          /* reserved                    */
   unsigned char    *alert;              /* Alert data                  */
} MS_ALERT_INDICATION;
```

Supplied Parameters

opcode
AP_ALERT_INDICATION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

alert_length
Length of the alert data.

alert
Pointer to the alert data. The data pointer is set to NULL, and the data is contiguous with (and begins immediately following) the VCB.

FP_NOTIFICATION

If an MDS-level application has been registered for a particular management services category and the status of a focal point for that category changes, then management services sends this verb signal to the application.

VCB Structure

```
typedef struct ms_fp_notification
{
   unsigned short   opcode;            /* Verb operation code        */
   unsigned char    reserv2;           /* reserved                   */
   unsigned char    format;            /* format                     */
   unsigned short   primary_rc;        /* Primary return code        */
   unsigned long    secondary_rc;      /* Secondary return code      */
   unsigned char    fp_routing;        /* Type of routing to focal pt */
   unsigned char    reserv1;           /* reserved                   */
   unsigned short   fp_data_length;    /* Length of incoming focal   */
                                       /* point data                 */
   unsigned char    *fp_data;          /* focal point data           */
} MS_FP_NOTIFICATION;
```

Supplied Parameters

opcode
AP_FP_NOTIFICATION

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

fp_routing
Type of routing that should be specified on the SEND_MDS_MU when sending a message to the focal point (AP_DEFAULT or AP_DIRECT).

fp_data_length
Length of focal point data.

fp_data
Focal point data containing a Focal Point Notification (X'E1') subvector and a Focal Point Identification (X'21') subvector. This data pointer is set to NULL, and the data is contiguous with (and begins immediately following) the VCB.

MDS_MU_RECEIVED

This verb indication is sent by management services to a registered MDS-level application when:

- An MDS_MU has been received from a peer MDS-level application
- An NMVT has been received, and
  - an appropriate NMVT-level application has not registered
  - The MDS-level application registered with a name that corresponds to the name carried within the management services major vector key in the incoming NMVT (management services performs the conversion from NMVT to MDS_MU).

VCB Structure

```
typedef struct ms_mds_mu_received
{
   unsigned short   opcode;           /* Verb operation code           */
   unsigned char    reserv2;          /* reserved                      */
   unsigned char    format;           /* format                        */
   unsigned short   primary_rc;       /* Primary return code           */
   unsigned long    secondary_rc;     /* Secondary return code         */
   unsigned char    first_message;    /* First message for curr MDS_MU */
   unsigned char    last_message;     /* Last message for curr MDS_MU  */
   unsigned char    pu_name[8];       /* Physical unit name            */
   unsigned char    reserv3[8];       /* reserved                      */
   unsigned short   mds_mu_length;    /* Length of incoming MDS_MU     */
   unsigned char    *mds_mu;          /* MDS_MU data                   */
} MS_MDS_MU_RECEIVED;
```

Supplied Parameters

opcode
AP_MDS_MU_RECEIVED

format
Identifies the format of the VCB. This field is set to zero to specify the version of the VCB listed above.

first_message
Flag indicating whether this is the first message for the MDS_MU (AP_YES or AP_NO). If the **max_rcv_size** specified in the **WinMSRegisterApplication** call is smaller than the length of the MDS_MU being delivered, the MDS_MU will be sent to the application in chunks.

last_message
Flag indicating whether this is the last message for the MDS_MU (AP_YES or AP_NO).

pu_name
Name of the physical unit from which the NMVT (which has been converted to an MDS_MU) originated. It is the responsibility of the application to respond to the incoming NMVT. The application uses SEND_MDS_MU to send the response. When sending responses the application must set the **pu_name** field of the SEND_MDS_MU to the **pu_name** supplied in the MDS_MU_RECEIVED signal. If the MDS_MU was received from the MDS level transport mechanism, the **pu_name** will be set to all binary zeros.

mds_mu_length
Length of MDS_MU portion included with the signal.

mds_mu
MDS_MU data. The data pointer is set to NULL, and the data is contiguous with (and begins immediately following) the VCB.

NMVT_RECEIVED

This verb signal is sent by management services to a registered NMVT-level application when an NMVT Is received from a remote node.

In routing incoming NMVTs, management services applies the following rules:

1. Try to route to an NMVT-level application registered with the major vector key carried on the incoming NMVT, else...
2. If the major vector key is one of X'8061' through X'8064', try to route to a registered NMVT-level AP_SPCF_KEYS application, else...
3. Try to route to an NMVT-level registered AP_ALL_KEYS application, else...
4. Try to route the NMVT (after conversion to an MDS_MU) to an MDS-level application, registered with the major vector key carried on the incoming NMVT, else...
5. If the major vector key is one of X'8061' through X'8064', try to route the NMVT (after conversion to an MDS_MU) to a registered MDS-level application, else...
6. Try to route (after conversion to an MDS_MU) to a registered AP_ALL_KEYS MDS-level application, else...
7. Negatively respond to the NMVT.

VCB Structure

```
typedef struct ms_nmvt_received
{
   unsigned short   opcode;            /* Verb operation code         */
   unsigned char    reserv2;           /* reserved                    */
   unsigned char    format;            /* format                      */
   unsigned short   primary_rc;        /* Primary return code         */
   unsigned long    secondary_rc;      /* Secondary return code       */
   unsigned char    pu_name[8];        /* Physical unit name          */
   unsigned char    reserv3[6];        /* reserved                    */
   unsigned short   nmvt_length;       /* Length of incoming NMVT     */
   unsigned char    *nmvt;             /* NMVT data                   */
} MS_NMVT_RECEIVED;
```

Supplied Parameters

opcode
AP_NMVT_RECEIVED

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

pu_name
Name of the physical unit from which the NMVT originated. It is the responsibility of the application to respond to the incoming NMVT. The application uses TRANSFER_MS_DATA to send the response. When sending responses, the application must set the **pu_name** field of the TRANSFER_MS_DATA to the **pu_name** supplied in the NMVT_RECEIVED signal.

nmvt_length
Length of NMVT data.

nmvt Full NMVT, containing management services major vector of the types specified on the REGISTER_NMVT_APPLICATION. This data pointer is set to NULL, and the data is contiguous with (and begins immediately following) the VCB.

SEND_MDS_MU

This verb is used by a MDS-level application to send network management data other than alerts using the **WinMS** entry point. If an error occurs during the sending of the MDS_MU to the destination application, the error is reported back to the origin application in one of two ways. If the error is detected at the local node, the application will be notified via the return codes of the SEND_MDS_MU response. If the error is detected at a remote node, the error is reported by means of an error MDS_MU transported in an MDS_MU_RECEIVED VCB. Management services can convert the outgoing MDS_MU to an NMVT if the destination node is to be reached via an SSCP-PU session. The application does not need to know the identity of its local node. If the application supplies 8 EBCDIC blanks in the **netid** or **nau** or both subfields of the origin location name subvector of the MDS Routing Information GDS variable, Personal Communications or Communications Server will supply the appropriate values. If an application does not fill in either the **netid** or **nau** but supplies fewer than 8 blanks, Personal Communications or Communications Server will return a secondary return code of AP_INVALID_MDS_MU_FORMAT.

VCB Structure

```
typedef struct ms_send_mds_mu
{
   unsigned short   opcode;            /* Verb operation code        */
   unsigned char    reserv2;           /* reserved                   */
   unsigned char    format;            /* format                     */
   unsigned short   primary_rc;        /* Primary return code        */
   unsigned long    secondary_rc;      /* Secondary return code      */
   unsigned char    options;           /* Verb options               */
   unsigned char    reserv3;           /* reserved                   */
   unsigned char    originator_id[8];  /* Originator ID              */
   unsigned char    pu_name[8];        /* Physical unit name         */
   unsigned char    reserv4[4];        /* reserved                   */
   unsigned short   dlen;              /* Length of data             */
   unsigned char    *dptr;             /* Data                       */
} MS_SEND_MDS_MU;
```

Supplied Parameters

opcode
AP_SEND_MDS_MU

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

options
Specifies optional processing on the data supplied on this verb. This parameter is a one-byte value, with individual bit settings indicating the options selected. If all options are specified, set this byte to zero.
Bit 0 is the most significant, and bit 7 is the least significant bit.
Bit 0: Adds Date/Time (X'01') subvector to the data if set to zero.
Bit 1: Adds Product Set ID (X'10') subvector to the data if set to zero. If the application supplies data that already contains a Product Set ID subvector, then management services adds Personal Communications or Communications Server’s Product Set ID subvector immediately before the existing one.
Bit 2: reserved.

Bit 3: Logs the data via the Personal Communications or Communications Server problem determination facility if set to zero.

Note: The following constants are provided in the management services header file that refer to bits 0, 1, and 3 specified above.
- SV_TIME_STAMP_SUBVECTOR
- SV_PRODUCT_SET_ID_SUBVECTOR
- SV_LOCAL_LOGGING

Bit 4: Specifies whether management services is to use default or direct routing to send the management services data to the destination application (AP_DEFAULT or AP_DIRECT).

Note: To set bit 4, use AP_DEFAULT or AP_DIRECT shifted appropriately (for example, AP_DIRECT<<3).

Bits 5–7: reserved.

originator_id
Name of component that issued the verb. This field is only used by management services when logging the SEND_MDS_MU.

pu_name
Name of the physical unit to send the data to. This should be set to either an 8-byte alphanumeric type-A EBCDIC string, padded to the right with EBCDIC spaces, or set to all binary zeros if no **pu_name** is specified. Applications using SEND_MDS_MU to respond to MDS_MU_RECEIVED indications that were converted from incoming NMVTs should specify the **pu_name** received in the MDS_MU_RECEIVED signal. MDS_MUs that are to be transported using the MDS transport facility should set the **pu_name** to all binary zeros.

dlen Length of data.

dptr Pointer to data. If this is set to NULL, management services assumes that the data is contiguous with (and begins immediately following) the VCB.

Returned Parameters

If the verb executes successfully, the Program management services returns the following parameter:

primary_rc
AP_OK

If the verb fails to execute because of a parameter error, the Program management services returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_INVALID_PU_NAME

AP_INVALID_MDS_MU_FORMAT
SV_INVALID_DATA_SIZE

If the verb fails to execute because of a state error, the Program management services returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
AP_SSCP_PU_SESSION_NOT_ACTIVE

If the verb does not execute because of a system error, the Program management services returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

TRANSFER_MS_DATA

This verb is used by NMVT-level applications to send unsolicited alerts and to respond to previously-received NMVT requests.

TRANSFER_MS_DATA is also used by MDS-level applications to send unsolicited alerts. This verb can be used by the application using the WinMS call.

VCB Structure

```
typedef struct ms_transfer_ms_data
{
   unsigned short   opcode;              /* Verb operation code      */
   unsigned char    data_type;           /* Data type supplied by app */
   unsigned char    format;              /* format                   */
   unsigned short   primary_rc;          /* Primary return code      */
   unsigned long    secondary_rc;        /* Secondary return code    */
   unsigned char    options;             /* Verb options             */
   unsigned char    reserv3;             /* reserved                 */
   unsigned char    originator_id[8];    /* Originator ID            */
   unsigned char    pu_name[8];          /* Physical unit name       */
   unsigned char    reserv4[4];          /* reserved                 */
   unsigned short   dlen;                /* Length of data           */
   unsigned char    *dptr;               /* Data                     */
} MS_TRANSFER_MS_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode
SV_TRANSFER_MS_DATA

data_type
Specifies the type of data enclosed. management services processes the data as described below. Allowed values:

SV_NMVT
The data contains a complete NMVT request unit. Management services converts the data to MDS_MU or CP_MSU format if the data contains an alert, and the alert is to be sent to an MDS-level or migration-level focal point. This is the type required when an application is responding to an NMVT_RECEIVED signal.

SV_ALERT_SUBVECTORS
The data contains management services subvectors in the SNA-defined format for an Alert major vector. Management services adds an NMVT header and an alert major vector header. Subsequently, management services converts the data to MDS_MU or CP_MSU format if the alert is to be sent to an MDS-level or migration-level focal point.

SV_USER_DEFINED
The data contains a complete NMVT request unit. Management services always logs the data, but does not send it.

SV_PDSTATS_SUBVECTORS
The data contains problem determination statistics. Management services always logs the data, and if an alert handler has been registered, then management services sends it the data within an ALERT_INDICATION.

format
Identifies the format of the VCB. Set this field to zero to specify the version of the VCB listed above.

options
Specifies optional processing on the data supplied on this verb. Note that management services processes the data primarily according to the **type** specified if there is any conflict between the **data_type** and the option specified. This parameter is a one-byte value, with individual bit settings indicating the options selected. If all options are specified, set this byte to zero.
Bit 0 is the most significant, and bit 7 is the least significant bit.
(Bits 1–3 are ignored if **data_type** is set to SV_USER_DEFINED.)
Bit 0: Adds Date/Time (X'01') subvector to the data if set to zero.
Bit 1: Adds Product Set ID (X'10') subvector to the data if set to zero. If the application supplies data that already contains a Product Set ID subvector, management services adds Personal Communications or Communications Server's Product Set ID subvector immediately before the existing one.
Bit 2: Sends the data on an SNA session if set to zero. Management services sends the data on the default SSCP-PU session if the data does not contain an alert. If the data contains an alert, management services sends the data on either an SSCP-PU session, a CP-CP session or an LU-LU session, depending on which type of session Personal Communications or Communications Server uses to transmit alerts to the alert focal point.
Bit 3: Logs the data via the Personal Communications or Communications Server problem determination facility if set to zero.

Note: The following constants are provided in the management services header file and they refer to the individual bits specified above.
- SV_TIME_STAMP_SUBVECTOR
- SV_PRODUCT_SET_ID_SUBVECTOR
- SV_SEND_ON_SESSION
- SV_LOCAL_LOGGING

Bits 4–7: reserved.

originator_id
Name of the component that issued the verb. This is an 8-byte string in a locally displayable character set. This field is only used by management services when logging the TRANSFER_MS_DATA.

pu_name
Name of the physical unit to send the data to. This should be set to either an 8-byte alphanumeric type-A EBCDIC string, padded to the right with EBCDIC spaces, or set to all binary zeros if no **pu_name** is specified. Applications using TRANSFER_MS_DATA to respond to NMVT_RECEIVED signals should specify the **pu_name** received in the NMVT_RECEIVED signal. The data contained in TRANSFER_MS_DATA signals of type SV_NMVT that do not specify a **pu_name** will be sent over the default PU session if available. TRANSFER_MS_DATA signals containing alerts should not specify a **pu_name** unless the application expressly wishes the alert data to be sent to a specific PU. This will bypass the normal management services alert routing algorithm.

dlen Length of data.

dptr Pointer to data. If this is set to NULL, then management services assumes that the data is contiguous with (and begins immediately following) the VCB.

Returned Parameters

If the verb executes successfully, management services returns the following parameter:

primary_rc
AP_OK

If the verb fails to execute because of a parameter error, management services returns the following parameters:

primary_rc
AP_PARAMETER_CHECK

secondary_rc
SV_INVALID_DATA_TYPE

SV_DATA_EXCEEDS_RU_SIZE
AP_INVALID_PU_NAME

If the verb fails to execute because of a state error, management services returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
SV_SSCP_PU_SESSION_NOT_ACTIVE

If the verb does not execute because of a system error, the Program management services returns the following parameter:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Part 3. Appendixes

Appendix A. IBM APPN MIB Tables

Table 4 gives details on implementing the tables from the IBM APPN management information block (MIB), as defined by RFC1593. The table defines:
- Node Operator Facility QUERY verb used to implement each MIB table
- Input parameter settings
- Any filtering operations required

(The mapping between the returned parameters and the MIB tables variables can be derived from the definition of the Node Operator Facility QUERY verbs). Personal Communications or Communications Server does not currently support the ibmappnNodePortDlcTraceTable and the ibmappnLsStatusTable MIB tables.

Table 4. Implementing Tables from IBM Management Information Block (MIB)

IBM MIB Table	Node Operator Facility Verb and MIB Table Variables	Input Parameter Settings
ibmappnNodePortTable	QUERY_PORT	**port_name** ibmappnNodePortName
ibmappnNodePortIpTable	(Note 1)	
ibmappnNodePortDlsTable	QUERY_PORT (select entries with **dlc_type** of AP_SDLC)	**port_name** ibmappnNodePortDlsName
ibmappnNodePortTrTable	QUERY_PORT	**port_name** ibmappnNodePortTrName
ibmappnNodeLsTable	QUERY_LS	**ls_name** ibmappnNodeLsName
ibmappnNodeLsIpTable	(Note 1)	
ibmappnNodeLsDlsTable	QUERY_LS (select entries with **dlc_type** of AP_SDLC)	**ls_name** ibmappnNodeLsDlsName
ibmappnNodeLsTrTable	QUERY_LS	**ls_name** ibmappnNodeLsTrName
ibmappnNnTopoRouteTable	QUERY_COS	**cos_name** ibmappnNnTopoRouteCos
ibmappnNnAdjNodeTable	QUERY_ADJACENT_NN	**adj_nncp_name** ibmappnNnAdjNodeAdjName
ibmappnNnTopologyTable	QUERY_NN_TOPOLOGY_NODE	**node_name** ibmappnNnNodeName **node_type** AP_LEARN_NODE **frsn** 0
ibmappnNnTgTopologyTable	QUERY_NN_TOPOLOGY_TG	**owner** ibmappnNnTgOwner **owner_type** AP_LEARN_NODE **dest** ibmappnNnTgDest **dest_type** AP_LEARN_NODE **tg_num** ibmappnNnTgNum **frsn** 0

Table 4. Implementing Tables from IBM Management Information Block (MIB) (continued)

IBM MIB Table	Node Operator Facility Verb and MIB Table Variables	Input Parameter Settings
ibmappnNnTopologyFRTable	QUERY_NN_TOPOLOGY_NODE	**node_name** ibmappnNnFRNode **node_type** AP_LEARN_NODE **frsn** ibmappnNnFRFrsn
ibmappnNnTgTopologyFRTable	QUERY_NN_TOPOLOGY_TG	**owner** ibmappnNnTgFROwner **owner_type** AP_LEARN_NODE **dest** ibmappnNnTgFRDest **dest_type** AP_LEARN_NODE **tg_num** ibmappnNnTgFRNum **frsn** ibmappnNnTgFRFrsn
ibmappnLocalTgTable	QUERY_LOCAL_TOPOLOGY	**dest** ibmappnLocalTGDest **dest_type** AP_LEARN_NODE **tg_num** ibmappnLocalTgNum
ibmappnLocalEnTable	QUERY_LOCAL_TOPOLOGY (select entries with unique **dest**) (Note 2)	**dest** ibmappnLocalEnName **dest_type** AP_END_NODE **dest_type** AP_LEARN_NODE
ibmappnLocalEnTgTable	QUERY_LOCAL_TOPOLOGY (Note 3)	**dest** ibmappnLocalEnTgOrigin **dest_type** AP_LEARN_NODE **tg_num** ibmappnLocalEnTgNum
ibmappnDirTable	QUERY_DIRECTORY_LU	**lu_name** ibmappnDirLuName
ibmappnCosModeTable	QUERY_MODE_TO_COS_MAPPING	**mode_name** ibmappnCosModeName
ibmappnCosNameTable	QUERY_COS	**cos_name** ibmappnCosName

Notes:

1. Personal Communications or Communications Server does not support IP as a DLC type.
2. Entries with the same **dest** are ordered consecutively by QUERY_LOCAL_TOPOLOGY.
3. The ibmappnLocalEnTgTable views TGs from the attached end node's perspective (that is, as a TG from the end node). However, a network node compliant with the current level of the APPN architecture only stores end node TG information for TGs between itself and directly attached end nodes. Therefore all the entries in this table have ibmappnLocalEnTgDest set to the name of the local node (ibmappnNodeCpName).

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

Index

Readers' Comments — We'd Like to Hear from You

Communications Server for Windows, Version 6.1
and
Personal Communications for Windows, Version 5.7
System Management Programming

Publication No. SC31-8480-06

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SC31-8480-06

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department CGMD / Bldg 500
P.O. Box 12195
Research Triangle Park, NC
27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8480-06

Cut or Fold
Along Line

Program Number: 5639–I70

Printed in U.S.A.

SC31-8480-06