

eNetwork Communications Server  
Version 6.0 for  
Windows NT and  
eNetwork Personal Communications  
Version 4.2  
for Windows 95 and Windows NT



# System Management Programming



eNetwork Communications Server  
Version 6.0 for  
Windows NT and  
eNetwork Personal Communications  
Version 4.2  
for Windows 95 and Windows NT



# System Management Programming

**Note**

Before using this information and the product it supports, read the information in "Appendix B. Notices" on page 635.

**Second Edition (July 1998)**

This edition applies to Version 6.0 of IBM eNetwork Communications Server for Windows NT, Version 4.2 of Personal Communications for Windows 95 and Windows NT, and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1989, 1997, 1998. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Tables</b> . . . . .	<b>vii</b>
<b>About This Book</b> . . . . .	<b>ix</b>
Who Should Read This Book . . . . .	ix
How to Use This Book . . . . .	x
Icons . . . . .	x
Conventions Used in This Book . . . . .	x
Where to Find More Information . . . . .	xii

## Part 1. Personal Communications and Communications Server Node Operator Facility . . . . . 1

<b>Chapter 1. Introduction</b> . . . . .	<b>3</b>
Purpose of the Document . . . . .	3
Personal Communications and Communications Server Node Operator Facility. . . . .	3
Entry Points . . . . .	3
Verb Control Blocks (VCBs) . . . . .	4
Writing Node Operator Facility (NOF) Programs . . . . .	4
Communications Server SNA API Client Support . . . . .	5
Verbs Supported by Communications Server and NOT by Personal Communications . . . . .	5

## Chapter 2. Overview of the Verbs in This Book . . . . . 7

How to Read Verb Descriptions . . . . .	7
Supplied Parameters . . . . .	7
Returned Parameters . . . . .	7
Common VCB Fields . . . . .	7
Verb Summary. . . . .	8
Node Configuration . . . . .	8
Activation and Deactivation . . . . .	9
Querying the Node . . . . .	10
Session Limit Verbs . . . . .	12
Unsolicited Indications . . . . .	12
Security Verbs . . . . .	13
APING Verbs . . . . .	14
CPI-C Verbs . . . . .	14
Attach Manager Verbs . . . . .	14
DLC Processes, Ports, and Link Stations . . . . .	14

## Chapter 3. Node Operator Facility Entry Points . . . . . 17

WinNOF() . . . . .	18
WinAsyncNOF() . . . . .	19
WinAsyncNOFEx() . . . . .	20
WinNOFCancelAsyncRequest() . . . . .	21
WinNOFCleanup() . . . . .	22
WinNOFStartup() . . . . .	23
WinNOFRegisterIndicationSink() . . . . .	24
WinNOFUnregisterIndicationSink() . . . . .	25
WinNOFGetIndication() . . . . .	26

## Chapter 4. Node Configuration Verbs 27

DEFINE_ADJACENT_NODE . . . . .	28
DEFINE_CN . . . . .	31
DEFINE_COS . . . . .	35
DEFINE_DEFAULTS . . . . .	41
DEFINE_DEFAULT_PU . . . . .	44
DEFINE_DLC . . . . .	46
DEFINE_DLUR_DEFAULTS . . . . .	50
DEFINE_DOWNSTREAM_LU . . . . .	52
DEFINE_DOWNSTREAM_LU_RANGE . . . . .	55
DEFINE_DSPU_TEMPLATE . . . . .	58
DEFINE_FOCAL_POINT . . . . .	61
DEFINE_INTERNAL_PU . . . . .	65
DEFINE_LOCAL_LU . . . . .	69
DEFINE_LS . . . . .	74
DEFINE_LU_0_TO_3 . . . . .	89
DEFINE_LU_0_TO_3_RANGE . . . . .	94
DEFINE_LU_POOL . . . . .	99
DEFINE_MODE . . . . .	101
DEFINE_PARTNER_LU . . . . .	107
DEFINE_PORT . . . . .	111
DEFINE_TP . . . . .	120
DELETE_ADJACENT_NODE . . . . .	124
DELETE_CN . . . . .	126
DELETE_COS . . . . .	128
DELETE_DLC . . . . .	130
DELETE_DOWNSTREAM_LU . . . . .	132
DELETE_DOWNSTREAM_LU_RANGE . . . . .	134
DELETE_DSPU_TEMPLATE . . . . .	136
DELETE_FOCAL_POINT . . . . .	139
DELETE_INTERNAL_PU . . . . .	141
DELETE_LOCAL_LU . . . . .	143
DELETE_LS . . . . .	145
DELETE_LU_0_TO_3 . . . . .	147
DELETE_LU_0_TO_3_RANGE . . . . .	149
DELETE_LU_POOL . . . . .	152
DELETE_MODE . . . . .	154
DELETE_PARTNER_LU . . . . .	156
DELETE_PORT . . . . .	158
DELETE_TP . . . . .	160

## Chapter 5. Activation and Deactivation Verbs . . . . . 163

START_DLC . . . . .	164
START_INTERNAL_PU . . . . .	166
START_LS . . . . .	168
START_PORT . . . . .	171
STOP_DLC . . . . .	173
STOP_INTERNAL_PU . . . . .	175
STOP_LS . . . . .	177
STOP_PORT . . . . .	179
ACTIVATE_SESSION . . . . .	181
DEACTIVATE_CONV_GROUP . . . . .	184
DEACTIVATE_SESSION . . . . .	186
PATH_SWITCH . . . . .	189

**Chapter 6. Query Verbs . . . . . 191**

QUERY_ADJACENT_NN . . . . .	192
QUERY_ADJACENT_NODE . . . . .	195
QUERY_CN . . . . .	199
QUERY_CN_PORT . . . . .	204
QUERY_CONVERSATION . . . . .	207
QUERY_COS . . . . .	211
QUERY_DEFAULT_PU . . . . .	214
QUERY_DEFAULTS . . . . .	216
QUERY_DIRECTORY_ENTRY . . . . .	218
QUERY_DIRECTORY_LU . . . . .	225
QUERY_DIRECTORY_STATS . . . . .	230
QUERY_DLC . . . . .	232
QUERY_DLUR_DEFAULTS . . . . .	238
QUERY_DLUR_LU . . . . .	240
QUERY_DLUR_PU . . . . .	244
QUERY_DLUS . . . . .	250
QUERY_DOWNSTREAM_LU . . . . .	254
QUERY_DOWNSTREAM_PU . . . . .	263
QUERY_DSPU_TEMPLATE . . . . .	268
QUERY_FOCAL_POINT . . . . .	272
QUERY_HPR_STATS . . . . .	277
QUERY_ISR_SESSION . . . . .	279
QUERY_LOCAL_LU . . . . .	290
QUERY_LOCAL_TOPOLOGY . . . . .	298
QUERY_LS . . . . .	303
QUERY_LS_EXCEPTION . . . . .	322
QUERY_LU_0_TO_3 . . . . .	327
QUERY_LU_POOL . . . . .	337
QUERY_MDS_APPLICATION . . . . .	341
QUERY_MDS_STATISTICS . . . . .	344
QUERY_MODE . . . . .	346
QUERY_MODE_DEFINITION . . . . .	352
QUERY_MODE_TO_COS_MAPPING . . . . .	357
QUERY_NMVT_APPLICATION . . . . .	360
QUERY_NN_TOPOLOGY_NODE . . . . .	363
QUERY_NN_TOPOLOGY_STATS . . . . .	369
QUERY_NN_TOPOLOGY_TG . . . . .	373
QUERY_NODE . . . . .	380
QUERY_PARTNER_LU . . . . .	392
QUERY_PARTNER_LU_DEFINITION . . . . .	399
QUERY_PORT . . . . .	404
QUERY_PU . . . . .	415
QUERY_RTP_CONNECTION . . . . .	421
QUERY_SESSION . . . . .	428
QUERY_SIGNED_ON_LIST . . . . .	436
QUERY_STATISTICS . . . . .	440
QUERY_TP . . . . .	442
QUERY_TP_DEFINITION . . . . .	446

**Chapter 7. Safe-Store Verbs . . . . . 451**

SAFE_STORE_TOPOLOGY . . . . .	452
SFS_ADJACENT_NN . . . . .	459
SFS_DIRECTORY . . . . .	463
SFS_NN_TOPOLOGY_NODE . . . . .	469
SFS_NN_TOPOLOGY_TG . . . . .	477

**Chapter 8. Session Limit Verbs . . . . . 485**

CHANGE_SESSION_LIMIT . . . . .	486
INITIALIZE_SESSION_LIMIT . . . . .	490

RESET_SESSION_LIMIT . . . . .	494
-------------------------------	-----

**Chapter 9. Node Operator Facility API Indications . . . . . 499**

DLC_INDICATION . . . . .	500
DLUR_LU_INDICATION . . . . .	501
DLUR_PU_INDICATION . . . . .	502
DLUS_INDICATION . . . . .	504
DOWNSTREAM_LU_INDICATION . . . . .	506
DOWNSTREAM_PU_INDICATION . . . . .	511
FOCAL_POINT_INDICATION . . . . .	514
ISR_INDICATION . . . . .	516
LOCAL_LU_INDICATION . . . . .	521
LOCAL_TOPOLOGY_INDICATION . . . . .	525
LS_INDICATION . . . . .	527
LU_0_TO_3_INDICATION . . . . .	532
MODE_INDICATION . . . . .	536
NN_TOPOLOGY_NODE_INDICATION . . . . .	538
NN_TOPOLOGY_TG_INDICATION . . . . .	540
PLU_INDICATION . . . . .	542
PORT_INDICATION . . . . .	544
PU_INDICATION . . . . .	546
REGISTER_INDICATION_SINK . . . . .	549
REGISTRATION_FAILURE . . . . .	551
RTP_INDICATION . . . . .	552
SESSION_INDICATION . . . . .	556
SESSION_FAILURE_INDICATION . . . . .	560
UNREGISTER_INDICATION_SINK . . . . .	562

**Chapter 10. Security Verbs . . . . . 565**

CONV_SECURITY_BYPASS . . . . .	566
CREATE_PASSWORD_SUBSTITUTE . . . . .	568
DEFINE_LU_LU_PASSWORD . . . . .	570
DEFINE_USERID_PASSWORD . . . . .	572
DELETE_LU_LU_PASSWORD . . . . .	574
DELETE_USERID_PASSWORD . . . . .	576
SIGN_OFF . . . . .	578

**Chapter 11. APING and CPI-C Verbs 581**

APING . . . . .	582
CPI-C Verbs . . . . .	586
DEFINE_CPIC_SIDE_INFO . . . . .	587
DELETE_CPIC_SIDE_INFO . . . . .	590
QUERY_CPIC_SIDE_INFO . . . . .	592

**Chapter 12. Attach Manager Verbs. . . 595**

DISABLE_ATTACH_MANAGER . . . . .	596
ENABLE_ATTACH_MANAGER . . . . .	597
QUERY_ATTACH_MANAGER . . . . .	598

---

**Part 2. Personal Communications and Communications Server Management Services API . . . . . 599**

**Chapter 13. Introduction to Management Services API . . . . . 601**

Management Services Verbs . . . . .	601
Entry Points . . . . .	601

Verb Control Blocks (VCB)	602
Writing Management Services (MS) Programs	602
SNA API Client Support	603

**Chapter 14. Management Services**

<b>Entry Points</b>	<b>605</b>
ACSSVC()	606
WinCSV()	607
WinMS()	608
WinMSCleanup()	609
WinMSStartup()	610
WinMSRegisterApplication()	611
WinMSUnregisterApplication()	614
WinMSGetIndication()	616

**Chapter 15. Management Services**

<b>Verbs</b>	<b>617</b>
TRANSFER_MS_DATA	618

MDS_MU_RECEIVED	621
SEND_MDS_MU	623
ALERT_INDICATION	626
FP_NOTIFICATION	627
NMVT_RECEIVED	628

<b>Appendix A. IBM APPN MIB Tables</b>	<b>631</b>
--	------------

<b>Appendix B. Notices</b>	<b>635</b>
----------------------------	------------

<b>Appendix C. Trademarks</b>	<b>637</b>
-------------------------------	------------

<b>Index</b>	<b>639</b>
--------------	------------

<b>Readers' Comments — We'd Like to Hear from You</b>	<b>643</b>
---	------------





---

## Tables

1. Header Files and Libraries for NOF	5	3. Header Files and Libraries for Management	
2. Port Types for DLC Types . . . . .	47	Services . . . . .	602



---

## About This Book

This book describes how to develop programs that use IBM eNetwork Communications Server for Windows NT and IBM eNetwork Personal Communications for Windows 95 and Windows NT .

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

IBM eNetwork Personal Communications for Windows 95 and Windows NT (referred to in this book 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 a 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 and Windows NT\*\*. 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 599, contains the following chapters:

- “Chapter 13. Introduction to Management Services API” on page 601, describes the management services API.
- “Chapter 14. Management Services Entry Points” on page 605, describes the entry points for the management services verbs.
- “Chapter 15. Management Services Verbs” on page 617, 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.

## Conventions Used in This Book

The following conventions are used throughout the Personal Communications or Communications Server library. Some of the conventions listed might not be used in this particular book.

## Text Conventions

---

<b>Bold</b>	Bold type indicates verbs, functions, and parameters that you can use in a program or at a command prompt. These values are case sensitive and should be entered exactly as they appear in the text.
<i>Italics</i>	Italic type indicates the following things: <ul style="list-style-type: none"><li>• A variable that you supply a value for.</li><li>• The names of window controls, such as lists, check boxed, entry fields, push buttons, and menu choices. They appear in the text as they appear in the window.</li><li>• Book titles.</li><li>• A letter is being used as a letter or a word is being used as a word. <b>Example:</b> When you see an <i>a</i>, make sure it is not supposed to be an <i>an</i>.</li></ul>
<b><i>Bold italics</i></b>	Bold italic type is used to emphasize a word.
UPPERCASE	Uppercase indicates constants, file names, keywords, and options that you can use in a program or at a command prompt. You can enter these values in uppercase or lowercase.
Double quotation marks	Double quotation marks indicate messages you see in a window. An example of this would be the messages that appear in the operator information area (OIA) of an emulator session.
Example type	Example type indicates information that you are instructed to type at a command prompt or in a window.

---

## 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, see *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 such as IBM Web Explorer, go to the following URL:

**<http://www.software.ibm.com/enetwork/commsserver/about/csnt.html>**



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

To view a specific book after Personal Communications 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 Personal Communications books are in BookManager format (BOO), which is viewable with the IBM Library Reader. If you do not have a copy of this program on your machine, you can install it from the eNetwork Personal Communications CD-ROM.

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.software.ibm.com/enetwork/pcomm/>**

# Part 1. Personal Communications and Communications Server Node Operator Facility

<b>Chapter 1. Introduction</b> . . . . .	3	DEFINE_DSPU_TEMPLATE . . . . .	58
Purpose of the Document . . . . .	3	DEFINE_FOCAL_POINT . . . . .	61
Personal Communications and Communications Server Node Operator Facility. . . . .	3	DEFINE_INTERNAL_PU . . . . .	65
Entry Points . . . . .	3	DEFINE_LOCAL_LU. . . . .	69
Verb Control Blocks (VCBs) . . . . .	4	DEFINE_LS . . . . .	74
Writing Node Operator Facility (NOF) Programs . . . . .	4	DEFINE_LU_0_TO_3 . . . . .	89
Communications Server SNA API Client Support . . . . .	5	DEFINE_LU_0_TO_3_RANGE . . . . .	94
Verbs Supported by Communications Server and <i>NOT</i> by Personal Communications . . . . .	5	DEFINE_LU_POOL . . . . .	99
		DEFINE_MODE . . . . .	101
<b>Chapter 2. Overview of the Verbs in This Book</b> . . . . .	7	DEFINE_PARTNER_LU . . . . .	107
How to Read Verb Descriptions . . . . .	7	DEFINE_PORT . . . . .	111
Supplied Parameters . . . . .	7	DEFINE_TP . . . . .	120
Returned Parameters . . . . .	7	DELETE_ADJACENT_NODE . . . . .	124
Return Codes . . . . .	7	DELETE_CN . . . . .	126
Additional Information . . . . .	7	DELETE_COS. . . . .	128
Common VCB Fields . . . . .	7	DELETE_DLC . . . . .	130
Verb Summary. . . . .	8	DELETE_DOWNSTREAM_LU . . . . .	132
Node Configuration . . . . .	8	DELETE_DOWNSTREAM_LU_RANGE . . . . .	134
Activation and Deactivation . . . . .	9	DELETE_DSPU_TEMPLATE . . . . .	136
Querying the Node . . . . .	10	DELETE_FOCAL_POINT . . . . .	139
Session Limit Verbs . . . . .	12	DELETE_INTERNAL_PU . . . . .	141
Unsolicited Indications . . . . .	12	DELETE_LOCAL_LU . . . . .	143
Security Verbs . . . . .	13	DELETE_LS . . . . .	145
APING Verbs . . . . .	14	DELETE_LU_0_TO_3 . . . . .	147
CPI-C Verbs . . . . .	14	DELETE_LU_0_TO_3_RANGE . . . . .	149
Attach Manager Verbs . . . . .	14	DELETE_LU_POOL . . . . .	152
DLC Processes, Ports, and Link Stations . . . . .	14	DELETE_MODE . . . . .	154
DLC Processes . . . . .	14	DELETE_PARTNER_LU . . . . .	156
Ports . . . . .	14	DELETE_PORT . . . . .	158
Link Stations . . . . .	15	DELETE_TP . . . . .	160
		<b>Chapter 5. Activation and Deactivation Verbs</b> . . . . .	163
<b>Chapter 3. Node Operator Facility Entry Points</b> . . . . .	17	START_DLC . . . . .	164
WinNOF() . . . . .	18	START_INTERNAL_PU. . . . .	166
WinAsyncNOF() . . . . .	19	START_LS . . . . .	168
WinAsyncNOFEx() . . . . .	20	START_PORT . . . . .	171
WinNOFCancelAsyncRequest() . . . . .	21	STOP_DLC. . . . .	173
WinNOFCleanup() . . . . .	22	STOP_INTERNAL_PU . . . . .	175
WinNOFStartup() . . . . .	23	STOP_LS . . . . .	177
WinNOFRegisterIndicationSink() . . . . .	24	STOP_PORT . . . . .	179
WinNOFUnregisterIndicationSink() . . . . .	25	ACTIVATE_SESSION . . . . .	181
WinNOFGetIndication() . . . . .	26	DEACTIVATE_CONV_GROUP . . . . .	184
		DEACTIVATE_SESSION . . . . .	186
<b>Chapter 4. Node Configuration Verbs</b> . . . . .	27	PATH_SWITCH . . . . .	189
DEFINE_ADJACENT_NODE . . . . .	28	<b>Chapter 6. Query Verbs</b> . . . . .	191
DEFINE_CN . . . . .	31	QUERY_ADJACENT_NN . . . . .	192
DEFINE_COS . . . . .	35	QUERY_ADJACENT_NODE . . . . .	195
DEFINE_DEFAULTS. . . . .	41	QUERY_CN . . . . .	199
DEFINE_DEFAULT_PU. . . . .	44	QUERY_CN_PORT . . . . .	204
DEFINE_DLC . . . . .	46	QUERY_CONVERSATION. . . . .	207
DEFINE_DLUR_DEFAULTS . . . . .	50	QUERY_COS . . . . .	211
DEFINE_DOWNSTREAM_LU. . . . .	52	QUERY_DEFAULT_PU . . . . .	214
DEFINE_DOWNSTREAM_LU_RANGE . . . . .	55	QUERY_DEFAULTS. . . . .	216

QUERY_DIRECTORY_ENTRY . . . . .	218
QUERY_DIRECTORY_LU . . . . .	225
QUERY_DIRECTORY_STATS . . . . .	230
QUERY_DLC . . . . .	232
QUERY_DLUR_DEFAULTS . . . . .	238
QUERY_DLUR_LU . . . . .	240
QUERY_DLUR_PU . . . . .	244
QUERY_DLUS . . . . .	250
QUERY_DOWNSTREAM_LU . . . . .	254
QUERY_DOWNSTREAM_PU . . . . .	263
QUERY_DSPU_TEMPLATE . . . . .	268
QUERY_FOCAL_POINT . . . . .	272
QUERY_HPR_STATS . . . . .	277
QUERY_ISR_SESSION . . . . .	279
QUERY_LOCAL_LU . . . . .	290
QUERY_LOCAL_TOPOLOGY . . . . .	298
QUERY_LS . . . . .	303
QUERY_LS_EXCEPTION . . . . .	322
QUERY_LU_0_TO_3 . . . . .	327
QUERY_LU_POOL . . . . .	337
QUERY_MDS_APPLICATION . . . . .	341
QUERY_MDS_STATISTICS . . . . .	344
QUERY_MODE . . . . .	346
QUERY_MODE_DEFINITION . . . . .	352
QUERY_MODE_TO_COS_MAPPING . . . . .	357
QUERY_NMVT_APPLICATION . . . . .	360
QUERY_NN_TOPOLOGY_NODE . . . . .	363
QUERY_NN_TOPOLOGY_STATS . . . . .	369
QUERY_NN_TOPOLOGY_TG . . . . .	373
QUERY_NODE . . . . .	380
QUERY_PARTNER_LU . . . . .	392
QUERY_PARTNER_LU_DEFINITION . . . . .	399
QUERY_PORT . . . . .	404
QUERY_PU . . . . .	415
QUERY_RTP_CONNECTION . . . . .	421
QUERY_SESSION . . . . .	428
QUERY_SIGNED_ON_LIST . . . . .	436
QUERY_STATISTICS . . . . .	440
QUERY_TP . . . . .	442
QUERY_TP_DEFINITION . . . . .	446
<b>Chapter 7. Safe-Store Verbs . . . . .</b>	<b>451</b>
SAFE_STORE_TOPOLOGY . . . . .	452
SFS_ADJACENT_NN . . . . .	459
SFS_DIRECTORY . . . . .	463
SFS_NN_TOPOLOGY_NODE . . . . .	469
SFS_NN_TOPOLOGY_TG . . . . .	477
<b>Chapter 8. Session Limit Verbs . . . . .</b>	<b>485</b>
CHANGE_SESSION_LIMIT . . . . .	486
INITIALIZE_SESSION_LIMIT . . . . .	490

RESET_SESSION_LIMIT . . . . .	494
-------------------------------	-----

**Chapter 9. Node Operator Facility API**

<b>Indications . . . . .</b>	<b>499</b>
DLC_INDICATION . . . . .	500
DLUR_LU_INDICATION . . . . .	501
DLUR_PU_INDICATION . . . . .	502
DLUS_INDICATION . . . . .	504
DOWNSTREAM_LU_INDICATION . . . . .	506
DOWNSTREAM_PU_INDICATION . . . . .	511
FOCAL_POINT_INDICATION . . . . .	514
ISR_INDICATION . . . . .	516
LOCAL_LU_INDICATION . . . . .	521
LOCAL_TOPOLOGY_INDICATION . . . . .	525
LS_INDICATION . . . . .	527
LU_0_TO_3_INDICATION . . . . .	532
MODE_INDICATION . . . . .	536
NN_TOPOLOGY_NODE_INDICATION . . . . .	538
NN_TOPOLOGY_TG_INDICATION . . . . .	540
PLU_INDICATION . . . . .	542
PORT_INDICATION . . . . .	544
PU_INDICATION . . . . .	546
REGISTER_INDICATION_SINK . . . . .	549
REGISTRATION_FAILURE . . . . .	551
RTP_INDICATION . . . . .	552
SESSION_INDICATION . . . . .	556
SESSION_FAILURE_INDICATION . . . . .	560
UNREGISTER_INDICATION_SINK . . . . .	562

**Chapter 10. Security Verbs**

<b>Chapter 10. Security Verbs . . . . .</b>	<b>565</b>
CONV_SECURITY_BYPASS . . . . .	566
CREATE_PASSWORD_SUBSTITUTE . . . . .	568
DEFINE_LU_LU_PASSWORD . . . . .	570
DEFINE_USERID_PASSWORD . . . . .	572
DELETE_LU_LU_PASSWORD . . . . .	574
DELETE_USERID_PASSWORD . . . . .	576
SIGN_OFF . . . . .	578

**Chapter 11. APING and CPI-C Verbs**

<b>Chapter 11. APING and CPI-C Verbs . . . . .</b>	<b>581</b>
APING . . . . .	582
CPI-C Verbs . . . . .	586
DEFINE_CPIC_SIDE_INFO . . . . .	587
DELETE_CPIC_SIDE_INFO . . . . .	590
QUERY_CPIC_SIDE_INFO . . . . .	592

**Chapter 12. Attach Manager Verbs**

<b>Chapter 12. Attach Manager Verbs . . . . .</b>	<b>595</b>
DISABLE_ATTACH_MANAGER . . . . .	596
ENABLE_ATTACH_MANAGER . . . . .	597
QUERY_ATTACH_MANAGER . . . . .	598



---

## Chapter 1. Introduction

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

---

### Purpose of the Document

The aim of the document 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 initialize the 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:

- WinAsyncNOF()
- WinAsyncNOFEx()
- WinNOFCancelAsyncRequest()
- WinNOFCleanup()
- WinNOFStartup()
- WinNOFRegisterIndicationSink()
- WinNOFUnregisterIndicationSink()

- WinNOFGetIndication()

See Chapter 3. Node Operator Facility Entry Points 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 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
WIN3.1	WINNOF.H	WINNOF.LIB	WINNOF.DLL
OS/2	APPC_C.H	APPC.LIB	APPC.DLL

---

## Communications Server SNA API Client Support



This information only applies to Communications Server .

Included with Communications Server are a set of clients for the Windows 95, Windows NT, Windows 3.1, 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 node operator facility. Specifically, **WINNOF** is the only API supported on the Windows clients (95, NT, 3.1). 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 only applies 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 631, 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 behaviour 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 following verb allows a management application to “ping” a remote LU in the network.

APING

## 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

See the *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications and Communications Server for Windows 95 and Windows NT.

## 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)
```

#### Parameter

##### Description

**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)
```

#### Parameter

##### Description

**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()**.

---

## 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);
```

#### Parameter

##### Description

#### 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()`.

---

## WinNOFCancelAsyncRequest()

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

### Syntax

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

#### Parameter

##### Description

#### **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()**.

---

## WinNOFCleanup()

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

### Syntax

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

### Returns

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

### Remarks

Use **WinNOFCleanup()** to indicate deregistration 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()**.

---

## 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);
```

#### Parameter

##### Description

##### 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()**.

---

## 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);
```

#### Parameter

##### Description

##### 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** and **WinNOFGetIndication**.

---

## 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);
```

#### Parameter

##### Description

##### **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** and **WinNOFGetIndication**.

---

## WinNOFGetIndication()

This allows the application to received unsolicited indications.

### Syntax

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

#### Parameter

##### Description

**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** and **WinNOFUnregisterIndicationSink**.



---

## 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 */
    unsigned char   indicator;     /* 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.

## DEFINE\_ADJACENT\_NODE

### **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 "." 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

## DEFINE\_ADJACENT\_NODE

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:

## DEFINE\_CN

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  $0.1mmm * 2^eeee$ , where the bit representation of the byte is  $eeeeemm$ . 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  $0.1mmm * 2 eeee$ , where the bit representation of the byte is  $eeeeemm$ . 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

## DEFINE\_CN

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              */
}
```

## DEFINE\_COS

```
        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  $0.1mm * 2^eeee$ , where the bit representation of the byte is  $eeeeemm$ . 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.

## DEFINE\_COS

### 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  $0.1mmm * 2^eeee$ , 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  $0.1mmm * 2^eeee$ , 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  $0.1mm * 2^eeee$ , where the bit representation of the byte is  $eeeemmm$ . 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:

## DEFINE\_COS

**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\_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; /* generic 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.

## DEFINE\_DEFAULTS

### **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\_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 617.

### 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\_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

**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.max\_activation\_attempts**

This field specifies whether the DLC only supports HPR links. This must be set to AP\_YES for HPR over IP links.

## DEFINE\_DLC

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 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.

### **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



## DEFINE\_DLUR\_DEFAULTS

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 only applies 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  */
    unsigned char   host_lu;           /* host LU                  */
    unsigned char   reserv2[6];         /* reserved                 */
} DOWNSTREAM_LU_DEF_DATA;
```

### Supplied Parameters

The application supplies the following parameters:

## DEFINE\_DOWNSTREAM\_LU

### 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 downstream 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 downstream 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

## DEFINE\_DOWNSTREAM\_LU

### **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 only applies 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 */
    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:

## DEFINE\_DOWNSTREAM\_LU\_RANGE

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

## DEFINE\_DOWNSTREAM\_LU\_RANGE

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 only applies 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



## DEFINE\_DSPU\_TEMPLATE

### **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 non-zero.

### **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.

### **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).

### **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 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.

## DEFINE\_DSPU\_TEMPLATE

### 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 the relevant START\_NODE parameter(s) 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.

## DEFINE\_FOCAL\_POINT

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.

## DEFINE\_FOCAL\_POINT

### **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

## DEFINE\_FOCAL\_POINT

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

## DEFINE\_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 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**

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

## DEFINE\_INTERNAL\_PU

**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 SDDL
    unsigned char  model_name[7];  /* LU model name
                                /* for SDDL
    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      */
```

## DEFINE\_LOCAL\_LU

```
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.

### 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 SDDL (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

## DEFINE\_LOCAL\_LU

be changed dynamically by re-issuing 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 SDDL, 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 re-issuing 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 under **def\_data.adj\_cp\_type**), 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 */
};
```

## DEFINE\_LS

```

    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 */

```

## DEFINE\_LS

```
        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 “\$ANYNETS” (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.

## DEFINE\_LS

### **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 **dest\_address** 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 pre-assigned 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**

Pre-assigned 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.

## DEFINE\_LS

### **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 non-switched 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\_LS

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 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.

**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 that 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

## DEFINE\_LS

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 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**

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\_activation\_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 **activation\_delay\_timer** supplied on DEFINE\_PORT.

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

## DEFINE\_LS

### **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

## DEFINE\_LS

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 DDDL */
    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;
```

## 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 DDDL (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 re-issuing 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 DDDL (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

## DEFINE\_LU\_0\_TO\_3

be changed dynamically by re-issuing 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

## DEFINE\_LU\_0\_TO\_3

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 DDDL */
    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 */
}
```

## DEFINE\_LU\_0\_TO\_3\_RANGE

```
    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:

## DEFINE\_LU\_0\_TO\_3\_RANGE

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 DDDL (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 re-issuing the verb. Changes will not come into effect until the LU is next closed and deactivated.

### **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.

### **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 SDDL (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

## DEFINE\_LU\_0\_TO\_3\_RANGE

### **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 to 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.

## DEFINE\_LU\_POOL

### 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_upper; /* max RU size upper bound  */
    unsigned char  receive_pacing_win; /* receive pacing window    */
    unsigned char  default_ru_size; /* default RU size to maximize */
    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 */
    unsigned short sessions; /* 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_lower; /* 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:

## DEFINE\_MODE

### **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.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\_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 activatable 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 autoactivated 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.

## DEFINE\_MODE

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

## DEFINE\_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**

## DEFINE\_MODE

### **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.

## DEFINE\_PARTNER\_LU

### **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 non-native (AnyNet) protocols only.

#### **AP\_NATIVE\_THEN\_NONNATIVE**

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

#### **AP\_NONNATIVE\_THEN\_NATIVE**

Try non-native (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:** Non-native routing is only meaningful when an AnyNet DLC is available to the Node Operator Facility, and there is an AnyNet link station defined. (See Defined\_LS).

### **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

## DEFINE\_PARTNER\_LU

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 unchanged.

**Note:** Issuing DEFINE\_PARTNER\_LU with a non-zero **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.

## DEFINE\_PARTNER\_LU

### **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 */
}
```

## DEFINE\_PORT

```
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 */
unsigned 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_support /* 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 reserv1; /* 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### def\_data.port\_number

Port number.

## DEFINE\_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.

### **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 **def\_data.tot\_link\_act\_lim - 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.

### **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 **def\_data.tot\_link\_act\_lim - 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.

### **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.

### **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 **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, 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.

**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\_activation\_attempts** supplied on DEFINE\_DLC.

**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 **activation\_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 **def\_data.retry\_flags**.

The value AP\_USE\_DEFAULTS results in the use of **activation\_delay\_timer** supplied on DEFINE\_DLC.

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

**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.

**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

## DEFINE\_PORT

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..

### **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 **def\_data.implicit\_dspu\_template**).

#### **AP\_NONE**

Local node will provide no services for this downstream PU.

### **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.

### **def\_data.act\_xid\_exchange\_limit**

Activation XID exchange limit.

### **def\_data.nonact\_xid\_exchange\_limit**

Non-activation XID exchange limit.

### **def\_data.ls\_xmit\_rcv\_cap**

Specifies the link station transmit/receive capability. 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).

### **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.

### **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.

**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.

**def\_data.dlc\_data.length**

Port address length.

**def\_data.dlc\_data.address**

Port address.

**def\_data.hpr\_dlc\_data.length**

HPR Port address length.

**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.

**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).

**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.

**def\_data.implicit\_hpr\_support**

Specifies whether HPR should be supported on implicit links (AP\_YES or AP\_NO).

**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.

**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**.

## DEFINE\_PORT

### **def\_data.discovery\_supported**

Specifies whether Discovery functions are to be performed on this port (AP\_YES or AP\_NO).

### **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.

### **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

## DEFINE\_PORT

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_rq;      /* 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       */
    unsigned char  parameters_supported /* parameters supported         */
    unsigned char  duplex_supported; /* duplex supported            */
    unsigned char  reserv3[9];       /* reserved                     */
    unsigned short tp_instance_limit; /* limit on currently active TP */
    unsigned short instances;        /* 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).

## DEFINE\_TP

### **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. See the Attach Manager in *Personal Communications Client/Server Communications Programming* for further details of how this is used.

### **tp\_chars.tp\_data.pathname**

Specifies the path and transaction program name.

### **tp\_chars.tp\_data.parameters**

Specifies the parameters for the transaction program.

### **tp\_chars.tp\_data.queued**

Specifies whether the transaction program will be queued.

### **tp\_chars.tp\_data.load\_type**

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 fq_lu_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 only applies 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 bitwise 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:

## DELETE\_DOWNSTREAM\_LU

**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 only applies 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 only applies 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\_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

## DELETE\_DSPU\_TEMPLATE

### **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 the relevant START\_NODE parameter(s) 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

## **DELETE\_DSPU\_TEMPLATE**

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 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.

#### 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.

## DELETE\_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\_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

## DELETE\_INTERNAL\_PU

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:

## DELETE\_LOCAL\_LU

**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:

## DELETE\_LS

**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:

## **DELETE\_LU\_0\_TO\_3**

**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

## 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.

## 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

## DELETE\_LU\_0\_TO\_3\_RANGE

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

## DELETE\_LU\_POOL

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



## DELETE\_MODE

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

## DELETE\_PARTNER\_LU

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

## DELETE\_PORT

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

## START\_DLC

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

## START\_LS

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

## START\_LS

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

## START\_PORT

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

## STOP\_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

## STOP\_INTERNAL\_PU

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 re-activated on demand by the local node.

##### AP\_REMOTE\_ACT

The link cannot be activated by the remote node. For a link

## STOP\_LS

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

## STOP\_PORT

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

## ACTIVATE\_SESSION

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 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:

## DEACTIVATE\_CONV\_GROUP

### 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



## DEACTIVATE\_SESSION

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

## DEACTIVATE\_SESSION

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:

## PATH\_SWITCH

**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.



See the *note pad* icon throughout this chapter for detailed information.

for detailed information.

---

## QUERY\_ADJACENT\_NN



This verb only applies 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) 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**.

## QUERY\_ADJACENT\_NN

### **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\_t dus**

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 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;
```

## QUERY\_ADJACENT\_NODE

```
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   fq_lu_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) 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.)

**cn\_data.det\_data.num\_act\_ports**

Dynamic value giving number of active ports on the connection network.

**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 non-zero.

**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\_data.adj\_lu\_def\_data.wildcard\_lu**

Indicates whether the LU name is defined as a wildcard..

**adjacent\_node\_lu\_data.adj\_lu\_def\_data.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.)

## QUERY\_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\_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;
```

## QUERY\_CN

```
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) 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  $0.1\text{mmmm} * 2\text{eeee}$ , where the bit representation of the byte is  $\text{eeeeemmm}$ . 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.

## QUERY\_CN

### **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  $0.1\text{mmmm} * 2\text{eeee}$ , where the bit representation of the byte is  $\text{eeeeemmm}$ . 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 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 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,

## QUERY\_CN\_PORT

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\_CN\_PORT 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 non-zero, 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

## 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

## QUERY\_CONVERSATION

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 */
    unsigned long  rejs;              /* number of route rejects */
    unsigned char  reserva[20];      /* reserved */
} COS_DATA;
```

### Supplied Parameters

The application supplies the following parameters:

**opcode**  
AP\_QUERY\_COS

## 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) 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 */
    unsigned char  /* 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.

## QUERY\_DEFAULT\_PU

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.

## QUERY\_DEFAULTS

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\_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 **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 */
}
```



## QUERY\_DIRECTORY\_ENTRY

```
    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 reserv1; /* reserved */
    unsigned short resource_type; /* Resource type */
    unsigned char description[RD_LEN]; /* resource description */
    unsigned char real_owing_cp_type; /* real owning CP type */
    unsigned char real_owing_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_owing_cp_type; /* real owning CP type */
    unsigned char real_owing_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;
```

## 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 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 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

## QUERY\_DIRECTORY\_ENTRY

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 non-zero, 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.)

## QUERY\_DIRECTORY\_ENTRY

### **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

## QUERY\_DIRECTORY\_ENTRY

### **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\_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\_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.

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

## QUERY\_DIRECTORY\_ENTRY

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        */
    unsigned char  server_name;      /* server name              */
    unsigned char  lu_owner_name[17]; /* network qualified        */
    unsigned char  lu_owner_name;    /* 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];
    unsigned char  apparent_lu_owner_name; /* apparent LU owner name */
    unsigned char  reserva[3];       /* reserved                  */
} DIRECTORY_LU_DETAIL;
```

## QUERY\_DIRECTORY\_LU

### 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) 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.

## QUERY\_DIRECTORY\_LU

### 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 only applies 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) 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.

## QUERY\_DLC

### 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. Personal Communications or Communications Server 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. Personal Communications or Communications Server 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

## QUERY\_DLC

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\_LU

**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 non-zero.

**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.) Location of LU. The only value returned is:

## QUERY\_DLUR\_DEFAULTS

### **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 the relevant START\_NODE parameter(s) 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 */
    unsigned char  session_active;   /* 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 */
}
```

```

        unsigned char   reserv1[27];      /* active          */
        unsigned char   rscv_len;        /* reserved        */
    } DLUR_LU_DETAIL;                  /* length of appended RSCV */

```

## 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) 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.

## QUERY\_DLUR\_LU

**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 */
}
```

## QUERY\_DLUR\_PU

```
    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 */
    unsigned char reserve3[3];        /* qualified CP name */
} 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) 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.

## QUERY\_DLUR\_PU

### 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.

## QUERY\_DLUR\_PU

### 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 retries 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 **d Lus\_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 **d Lus\_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  d Lus_name[17];   /* fully qualified DLUS name    */
} QUERY_DLUS;

typedef struct d Lus_data
{
    unsigned short overlay_size;     /* size of this entry          */
    unsigned char  d Lus_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                   */
}
```



## QUERY\_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 **dplus\_name** specified (see 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 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.

### dplus\_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

## QUERY\_DLUS

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 only applies 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 */
    unsigned char  nau_address;       /* downstream node */
    unsigned char  lu_sscp_sess_active; /* NAU address */
    unsigned char  plu_sess_active;   /* Is LU-SSCP session active */
    unsigned char  plu_sess_active;   /* Is PLU-SLU session active */
} DOWNSTREAM_LU_SUMMARY
```

## QUERY\_DOWNSTREAM\_LU

```
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_pu_name[8]; /* Determined host PU name */
    unsigned char reserva[4]; /* reserved */
} DOWNSTREAM_LU_DET_DATA;

typedef struct downstream_lu_def_data
{
    unsigned char description[RD_LEN]; /* resource description */
    unsigned char nau_address; /* NAU address */
    unsigned char dspu_name[8]; /* Downstream PU name */
    unsigned char host_pu_name; /* host LU or pool name */
    unsigned char allow_timeout; /* Allow timeout of host LU? */
    unsigned char delayed_logon; /* Allow delayed logo 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:

## QUERY\_DOWNSTREAM\_LU

### **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) 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).

## QUERY\_DOWNSTREAM\_LU

### **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.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.

## QUERY\_DOWNSTREAM\_LU

**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.lu\_sscp\_pacing\_type**

Receive pacing in use on the LU-SSCP session. This takes the value AP\_NONE.

**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, Personal Communications or Communications Server 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 Personal Communications or Communications Server 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 Personal Communications or Communications Server should delay connecting the downstream LU to the host LU until the first

## QUERY\_DOWNSTREAM\_LU

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 only applies 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 */
}
```

## QUERY\_DOWNSTREAM\_PU

```

/* 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) 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**.

## QUERY\_DOWNSTREAM\_PU

### **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



## QUERY\_DOWNSTREAM\_PU

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 only applies 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  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) 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

## QUERY\_DSPU\_TEMPLATE

### **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.

### **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).

### **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 is sent to the downstream LU (AP\_YES or AP\_NO).

### **dslu\_template\_data.dslu\_template.host\_lu\_name**

Name of the host LU or host LU pool that all the downstream LUs within

## QUERY\_DSPU\_TEMPLATE

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 the relevant START\_NODE parameter(s) 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) 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.

## QUERY\_FOCAL\_POINT

### **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.bkup\_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.

**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

## QUERY\_FOCAL\_POINT

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 only applies to Communications Server .

QUERY\_HPR\_STATS returns statistics describing the HPR performance of the node. QUERY\_HPT\_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 origin  */
    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 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.

## QUERY\_HPR\_STATS

### **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 only applies 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;
```

## QUERY\_ISR\_SESSION

```
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 */
}

```

## QUERY\_ISR\_SESSION

```
    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) 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.pcid\_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.

## QUERY\_ISR\_SESSION

### **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.

## QUERY\_ISR\_SESSION

- 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

## QUERY\_ISR\_SESSION

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 DEFINE\_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 DEFINE\_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 DEFINE\_ISR\_STATS.

## QUERY\_ISR\_SESSION

### **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.

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 the relevant START\_NODE parameter(s) 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          */
```



## QUERY\_LOCAL\_LU

```
    unsigned char  lu_name[8];          /* LU name */
    LOCAL_LU_DEF_DATA def_data;        /* defined data */
    LOCAL_LU_DEF_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 SDDL */
    unsigned char  model_name[8];       /* LU model name for SDDL */
    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

## QUERY\_LOCAL\_LU

```
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.

## QUERY\_LOCAL\_LU

### **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 (off-line).

### **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.

**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 SDDL (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.

**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 SDDL, 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 reissuin 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.

## QUERY\_LOCAL\_LU

**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 */
}
```



## QUERY\_LOCAL\_TOPOLOGY

```
    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_tg;       /* branch link type    */
    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 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 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

## QUERY\_LOCAL\_TOPOLOGY

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

## QUERY\_LOCAL\_TOPOLOGY

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:

## QUERY\_LOCAL\_TOPOLOGY

### 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 */
}
```

## QUERY\_LS

```
        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];
    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_labe; /* reverse ANR label */
    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 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 reserv1;
    /* reserved */
    unsigned char address[MAX_LINK_ADDR_LEN];
    /* address */
} LINK_ADDRESS;

```

## QUERY\_LS

```
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 */
}
```



## QUERY\_LS

```
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) 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.

## QUERY\_LS

### 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. Personal Communications or Communications Server 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

## QUERY\_LS

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. Personal Communications or Communications Server 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.

## QUERY\_LS

### **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.

## QUERY\_LS

### **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.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**

Pre-assigned 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 pre-assigned 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.

## QUERY\_LS

### **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.

**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.

## QUERY\_LS

### **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 **ls\_detail.def\_data.dlus\_name** and **ls\_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 **ls\_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.

**Is\_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.

**Is\_detail.def\_data.convention\_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.

**Is\_detail.def\_data.convention\_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.

**Is\_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.

## QUERY\_LS

### 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\_activation\_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 **activation\_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. 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\_cp\_is\_brnn

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 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_index;  /* 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 */
}
```



## QUERY\_LS\_EXCEPTION

```
    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 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.

### 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:

## QUERY\_LS\_EXCEPTION

**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 is incremented 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. Personal Communications or Communications Server 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\_address.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

## QUERY\_LS\_EXCEPTION

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* 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                    */
}
```

## QUERY\_LU\_0\_TO\_3

```
    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    */
    unsigned short window_size;        /* 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];          /* SSCP ID                   */
    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) 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.

## QUERY\_LU\_0\_TO\_3

### **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:

## QUERY\_LU\_0\_TO\_3

### 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.

## QUERY\_LU\_0\_TO\_3

- 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.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.app\_type**  
Reserved.
- lu\_0\_to\_3\_detail.lu\_0\_to\_3\_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.

## QUERY\_LU\_0\_TO\_3

### **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.

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:

## QUERY\_LU\_POOL

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 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 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 auto-activable, and whose connection is free. This count is regardless of the LU **model\_type**, **model\_name**, and the DDDL 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.num\_active\_lus**

The number of LUs in the specified pool that have active LU-SSCP sessions.

**lu\_pool\_detail.num\_avail\_lus**

The number of LUs in the specified pool that have available LU-SSCP sessions.

**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).

## QUERY\_LU\_POOL

### **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 617.

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.

## QUERY\_MDS\_APPLICATION

### **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) 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.

## QUERY\_MDS\_APPLICATION

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 */
}
```



## QUERY\_MODE

```
    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 */
    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 */
    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.

## QUERY\_MODE

### 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 parameter) 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.

## QUERY\_MODE

### **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_upper; /* max RU size upper bound */
    unsigned char  receive_pacing_win; /* receive pacing window */
}
```

## QUERY\_MODE\_DEFINITION

```
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) 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.

## QUERY\_MODE\_DEFINITION

### **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.



## QUERY\_MODE\_DEFINITION

### **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—32 767

### **mode\_def\_detail.mode\_chars.min\_conwin\_src**

Minimum number of contention winner sessions activatable 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

## QUERY\_MODE\_DEFINITION

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.

## QUERY\_MODE\_TO\_COS\_MAPPING

### **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) 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).

## QUERY\_MODE\_TO\_COS\_MAPPING

**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 617 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  */
    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) 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

## QUERY\_NMVT\_APPLICATION

services vector keys it will accept. For more information on NMVT application registration see “Chapter 15. Management Services Verbs” on page 617 .

### **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 617.

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 only applies 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 */
}
```

## QUERY\_NN\_TOPOLOGY\_NODE

```
    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 rsrn;              /* 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 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 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).

## QUERY\_NN\_TOPOLOGY\_NODE

### **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 end-point sessions has reached the maximum specified.

#### **AP\_IRR\_DEPLETED**

The number of ISR sessions has reached the maximum.

## QUERY\_NN\_TOPOLOGY\_NODE

### 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:

## QUERY\_NN\_TOPOLOGY\_NODE

**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 only applies 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 short 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;
```

## 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

## QUERY\_NN\_TOPOLOGY\_STATS

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 only applies 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 */
}
```

## QUERY\_NN\_TOPOLOGY\_TG

```
        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 reserv1; /* 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 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 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**.

## QUERY\_NN\_TOPOLOGY\_TG

### **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

## QUERY\_NN\_TOPOLOGY\_TG

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.

## QUERY\_NN\_TOPOLOGY\_TG

### **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.

### **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**

Is this TG an intersubnetwork TG?

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).

### **branch\_tg**

Specifies whether the TG is a branch TG.



## QUERY\_NN\_TOPOLOGY\_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_frns;           /* NN flow reduction            */
    unsigned long  nn_rsn;            /* Resource sequence number     */
    unsigned short def_ls_good_xids; /* Good XIDs for defined        */
    unsigned short def_ls_bad_xids; /* Bad XIDs for defined         */
    unsigned short dyn_ls_good_xids; /* Good XIDs for dynamic        */
    unsigned short dyn_ls_bad_xids; /* Bad XIDs for dynamic         */
    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                     */
}
```

## QUERY\_NODE

```

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          */
}

```

## QUERY\_NODE

```
unsigned long secondary_rc; /* secondary return code */
CP_CREATE_PARAMS 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_frns; /* 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_PARAMS 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_frns; /* 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 */
}
```

## QUERY\_NODE

```
                /* 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.

## QUERY\_NODE

### **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 551 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.

## QUERY\_NODE

### **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 uncongested to congested if the number of ISR sessions exceeds the upper threshold. The node state changes back to uncongested 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 non-native (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 non-native (AnyNet) protocols.

**AP\_NONNATIVE\_THEN\_NATIVE**

Try non-native (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 non-native (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:

## QUERY\_NODE

**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

## QUERY\_NODE

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   plu_alias[8];     /* partner LU alias             */
    unsigned char   fqplu_name[17];   /* fully qualified partner      */
    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             */
}
```

## QUERY\_PARTNER\_LU

```
    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:

### 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

## QUERY\_PARTNER\_LU

all zeros) specified (see 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 **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.

## QUERY\_PARTNER\_LU

### **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.implicit**

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 non-native (Anynet) protocols, and if the partner LU cannot be located, then retry session activation using non-native (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 non-native 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:

## QUERY\_PARTNER\_LU

**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 */
```

## QUERY\_PARTNER\_LU\_DEFINITION

```
    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 */
                                        /* already verified accepted */
    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) 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.

## QUERY\_PARTNER\_LU\_DEFINITION

### 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.)

## QUERY\_PARTNER\_LU\_DEFINITION

### **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.

### **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 non-native (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 non-native (AnyNet) protocols.

#### **AP\_NONNATIVE\_THEN\_NATIVE**

Try non-native (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:** Non-native routing is only meaningful when an AnyNet DLC is available to the Node Operator Facility, and there is an AnyNet link station defined.



## QUERY\_PARTNER\_LU\_DEFINITION

### **partner\_lu\_def\_detail.plu\_chars.max\_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\_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 retries */
                                           /* 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 */

```

## QUERY\_PORT

```

/* 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) 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.

## QUERY\_PORT

### **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. Personal Communications or Communications Server 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.def\_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

## QUERY\_PORT

### **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..



**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 **def\_data.implicit\_dspu\_template**).

**AP\_NONE**

Local node will provide no services for this downstream PU.

**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

## QUERY\_PORT

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_activation_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 `activation_delay_timer` supplied on `DEFINE_PORT`.

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

### **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.

### **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.

### **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.

## QUERY\_PORT

### **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 rcv pacing window size */
    unsigned short cur_rcv_pac_win;  /* current receive pacing */
}
```

## QUERY\_PU

```

/* 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) 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:

## QUERY\_PU

### 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.

**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:

**QUERY\_PU**

**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               */
}
```

## QUERY\_RTP\_CONNECTION

```
    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**.

## QUERY\_RTP\_CONNECTION

### **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).

## QUERY\_RTP\_CONNECTION

- 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



## QUERY\_RTP\_CONNECTION

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
```

## QUERY\_SESSION

```

    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;

```

## QUERY\_SESSION

**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 parameter) 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.

## QUERY\_SESSION

### **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.

## QUERY\_SESSION

### **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 parameter) 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

## QUERY\_SIGNED\_ON\_LIST

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** 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

## QUERY\_SIGNED\_ON\_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

---

## 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 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 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

## QUERY\_TP

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.instance.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\_chars.tp\_data.pathname**

Specifies the path and transaction program name.

**tp\_chars.tp\_data.parameters**

Specifies the parameters for the transaction program.

**tp\_chars.tp\_data.queued**

Specifies whether the transaction program will be queued.

**tp\_chars.tp\_data.load\_type**

Specifies how the transaction program will be loaded.

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_rq;      /* security support             */
}
```

## QUERY\_TP\_DEFINITION

```
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.

## QUERY\_TP\_DEFINITION

The **tp\_name** specified (see 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 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.

## QUERY\_TP\_DEFINITION

### **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.

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*/
}
```

## SAFE\_STORE\_TOPOLOGY

```
    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 */
} RESOURCE_INDEX;
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 */
    RESOURCE_INDEX index;             /* index of appended resource */
} 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.

## SAFE\_STORE\_TOPOLOGY

### **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 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 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 bitwise 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

## SAFE\_STORE\_TOPOLOGY

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 destination 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 non-zero, 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\_TOPOLOGY

### **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\_node\_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\_name**

Network fully qualified resource 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 ignored if **list\_options** is set to AP\_FIRST\_IN\_LIST.

**resource\_type**

This bit field controls the topology data to be stored. Any combination of the following values may be bitwise ORed together in this field:

## SAFE\_STORE\_TOPOLOGY

**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 the relevant START\_NODE parameter(s) 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 **resource\_types** and **index** specified (see 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 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**

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\_t dus**

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.

## SFS\_ADJACENT\_NN

### **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**

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 the relevant START\_NODE parameter(s) was 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 located 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;
```

## 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_owing_cp_type;    /* real owning CP type */
    unsigned char real_owing_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\_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 **resource\_name** and **resource\_type** specified (see 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 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 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

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 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.

**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

## SFS\_DIRECTORY

### **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 resorte = 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 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 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 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. See one of the following:

AP\_NNCP\_RESOURCE  
 AP\_ENCP\_RESOURCE  
 AP\_LU\_RESOURCE

**directory\_entry\_summary.real\_owing\_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\_owing\_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:

## SFS\_DIRECTORY

**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 the relevant START\_NODE parameter(s) was 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 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 non-zero 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        */
}
```

## SFS\_NN\_TOPOLOGY\_NODE

```
    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 rsrn; /* 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 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 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 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 non-zero, 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.)

## SFS\_NN\_TOPOLOGY\_NODE

### **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 end-point sessions has reached the maximum specified.

#### **AP QUIESCING**

A STOP\_NODE of type AP\_QUIESCENCE or AP\_QUIESCENCE\_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.

## SFS\_NN\_TOPOLOGY\_NODE

### **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 end-point sessions has reached the maximum specified.

#### **AP QUIESCING**

A STOP\_NODE of type AP\_QUIESCING or AP\_QUIESCING\_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.



## SFS\_NN\_TOPOLOGY\_NODE

### 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 non-zero, 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 the relevant START\_NODE parameter(s) 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

## SFS\_NN\_TOPOLOGY\_NODE

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\_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). 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 non-zero 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        */
}
```

## SFS\_NN\_TOPOLOGY\_TG

```
    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 rs_n; /* 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 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 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 non-zero, 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**.

**topology\_tg\_detail.overlay\_size**

The number of bytes in this entry, and hence the offset to the next entry returned (if any).

**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.)

**topology\_tg\_detail.owner\_type**

Type of the node. It is one of the following:

AP\_NETWORK\_NODE

AP\_VRN

**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.

**topology\_tg\_detail.dest\_type**

Type of the node. It is one of the following:

AP\_NETWORK\_NODE

AP\_VRN

**topology\_tg\_detail.tg\_num**

The number associated with the TG.

**topology\_tg\_detail.frsn**

The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

**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.

**topology\_tg\_detail.dlc\_data.length**

The address length.

**topology\_tg\_detail.dlc\_data.address**

The address.

**topology\_tg\_detail.rsn**

The Resource Sequence Number. This is assigned by the network node that owns this resource.

**topology\_tg\_detail.status**

This field specifies the status of theTG. 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

**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

## 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**.

### **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.

### **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.)

### **topology\_tg\_detail.owner\_type**

Type of the node that owns the TG. It is one of the following:

AP\_NETWORK\_NODE  
AP\_VRN

### **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.

### **topology\_tg\_detail.dest\_type**

Type of the node. It is one of the following:

AP\_NETWORK\_NODE  
AP\_VRN

### **topology\_tg\_detail.tg\_num**

The number associated with the TG.



**topology\_tg\_detail.frsn**

The Flow Reduction Sequence Number. This indicates the last time that the resource was updated at the local node.

**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.

**topology\_tg\_detail.dlc\_data.length**

The address length.

**topology\_tg\_detail.dlc\_data.address**

The address.

**topology\_tg\_detail.rsn**

The Resource Sequence Number. This is assigned by the network node that owns this resource.

**topology\_tg\_detail.status**

This field specifies the status of theTG. 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

**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 the relevant START\_NODE parameter(s) 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:

## SFS\_NN\_TOPOLOGY\_TG

**primary\_rc**  
AP\_INVALID\_VERB

If the verb does not execute because the 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  fqp1u_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:

## CHANGE\_SESSION\_LIMIT

**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:

## CHANGE\_SESSION\_LIMIT

### **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



## INITIALIZE\_SESSION\_LIMIT

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

## INITIALIZE\_SESSION\_LIMIT

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:

## INITIALIZE\_SESSION\_LIMIT

### **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

## RESET\_SESSION\_LIMIT

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

## RESET\_SESSION\_LIMIT

### **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

## RESET\_SESSION\_LIMIT

### 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

#### 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.

## DLUR\_PU\_INDICATION

**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.

### **plu\_location**

The location of the PU. This can be one of the following:

AP\_INTERNAL  
AP\_DOWNSTREAM

### **dlur\_pu\_detail.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.

**pu\_id** 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.

### **dlur\_pu\_detail.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

## DLUS\_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 only applies 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   ds_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   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.

## DOWNSTREAM\_LU\_\_INDICATION

### 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.

## DOWNSTREAM\_LU\_\_INDICATION

### **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.

## DOWNSTREAM\_LU\_INDICATION

- 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 only applies 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:

## DOWNSTREAM\_PU\_\_INDICATION

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.

## DOWNSTREAM\_PU\_\_INDICATION

**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  */
    unsigned char   fp_type;          /* type of current focal    */
    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).



## FOCAL\_POINT\_\_INDICATION

### **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 only applies 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    */
                                        /* sense data                */
    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   */
}
```

## ISR\_INDICATION

```
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

### 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.

### deactivate

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.pcid\_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.

### 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.

## ISR\_INDICATION

### **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

## ISR\_INDICATION

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

## **ISR\_INDICATION**

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

## 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 non-zero 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.

## LOCAL\_LU\_\_INDICATION

### **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.)

## LOCAL\_TOPOLOGY\_INDICATION

### **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. 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.

---

## 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

## 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\_INDICATION

### **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\_brnnt**

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:

## LU\_0\_TO\_3\_INDICATION

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\_0\_TO\_3\_INDICATION

### **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

## MODE\_INDICATION

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 only applies 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\_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

#### 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



## NN\_TOPOLOGY\_NODE\_INDICATION

### **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 only applies 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

#### 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

## NN\_TOPOLOGY\_TG\_INDICATION

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**

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.

---

## 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  fqp_lu_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.

## **PORT\_INDICATION**

### **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\_INDICATION

### **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.

---

## REGISTER\_INDICATION\_SINK

REGISTER\_INDICATION\_SINK registers a process and queue to which indications should be sent.

The **orig\_verb\_data** in the verb\_signal header of REGISTER\_INDICATION\_SINK is returned in the verb\_signal header of any indications received.

### VCB Structure

```
typedef struct port_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 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                    */
} REGISTER_INDICATION_SINK;
```

### Parameters

**opcode**

AP\_REGISTER\_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 receiving process.

**queue\_id**

Queue ID where receiving process indications should be sent.

**indication\_opcode**

Opcode of the indication that is to be returned whenever generated, once the indication sink has been registered.

### 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

## REGISTER\_INDICATION\_SINK

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 the relevant START\_NODE parameter(s) were not sent, 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

---

## 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

#### 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\_INDICATION

### **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\_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 */
}
```

## SESSION\_INDICATION

```
    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

## SESSION\_INDICATION

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\_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.

---

## 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

#### **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.

## SESSION\_FAILURE\_INDICATION

### **sense\_data**

The sense data detailing the cause of the session deactivation.

---

## 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\_datain** the **verb\_signal** header of UNREGISTER\_INDICATION\_SINK is removed.

### VCB Structure

```
typedef struct port_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 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                    */
} REGISTER_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



## UNREGISTER\_INDICATION\_SINK

### 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 the relevant START\_NODE parameter(s) were not sent, 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      */
    unsigned char  reserv3[3];       /* reserved                  */
} DEFINE_LU_LU_PASSWORD;
```

### 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

## CONV\_SECURITY\_BYPASS

### **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.

## CREATE\_PASSWORD\_SUBSTITUTE

### **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 */
                                     /* resource description      */
    unsigned char   description[RD_LEN];
    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-byte s 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  */
    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

## DELETE\_LU\_LU\_PASSWORD

**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\_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\_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



## SIGN\_OFF

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:

### user\_id

The user ID.

### all\_profiles

Total number of entries that could have been returned. This can be higher than **num\_entries**.

**profile** 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

## SIGN\_OFF

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**

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

## APING

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



## 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. See *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications or Communications Server .

### VCB Structure

```
typedef struct define_cplic_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.

## DEFINE\_CPIC\_SIDE\_INFO

### **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. See the *CPI-C Reference* for more information about the CPI-C support provided by Personal Communications or Communications Server .

### VCB Structure

```
typedef struct delete_cplic_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

## **DELETE\_CPIC\_SIDE\_INFO**

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_cplic_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.

## QUERY\_CPIC\_SIDE\_INFO

### **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

<b>Services API</b> . . . . .	601
Management Services Verbs . . . . .	601
Entry Points . . . . .	601
Verb Control Blocks (VCB). . . . .	602
Writing Management Services (MS) Programs	602
SNA API Client Support. . . . .	603

### Chapter 14. Management Services Entry Points

ACSSVC() . . . . .	606
WinCSV() . . . . .	607
WinMS(). . . . .	608
WinMSCleanup() . . . . .	609

WinMSStartup() . . . . .	610
WinMSRegisterApplication() . . . . .	611
WinMSUnregisterApplication(). . . . .	614
WinMSGetIndication() . . . . .	616

### Chapter 15. Management Services Verbs

TRANSFER_MS_DATA . . . . .	618
MDS_MU_RECEIVED . . . . .	621
SEND_MDS_MU . . . . .	623
ALERT_INDICATION . . . . .	626
FP_NOTIFICATION . . . . .	627
NMVT_RECEIVED . . . . .	628



---

## Chapter 13. Introduction to Management Services API

This part 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:

- ACSSVC()
- WinMS()
- WinAsyncMS()
- WinAsyncMSEx()
- WinCSV()
- WinMSCancelAsyncRequest()
- WinMSCleanup()
- WinMSStartup()

See “Chapter 14. Management Services Entry Points” on page 605 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 on page 603 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
WINNT & WIN95	WINMS.H	WINMS32.LIB	WINMS32.DLL
WIN3.1	WINCSV.H	WINCSV.LIB	WINCSV.DLL
OS/2	ACSSVCC.H	ACSSVC.LIB	ACSSVC.DLL

---

## SNA API Client Support

Included with Communications Server are a set of clients for the Windows 95, Windows NT, Windows 3.1, 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 95 and NT clients, see “WinMS()” on page 608 for more information.
- **WINCSV** is only supported on Windows 3.1 clients, see “WinCSV()” on page 607 for more information.
- **ACSSVC** is only supported on SNA API OS/2 clients, see “ACSSVC()” on page 606 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.

---

## ACSSVC()



This is the only entry point supported for SNA API OS/2 clients.

This provides a synchronous entry point for issuing the following management services API verbs on the OS/2 SNA API Clients.

### Syntax

```
void ACSSVC (long);
```

Input is a verb control block pointer.

### Returns

Check the primary and secondary return codes for returned values.

---

## WinCSV()



This is the only entry point supported for Windows 3.1 clients.

This function provides a synchronous entry point for the CSV API.

### Syntax

```
void WINAPI WinCSV(long vcb)
```

#### Parameter

##### Description

**vcb** Pointer to verb control block.

### Returns

No return value. The **primary\_rc** and **secondary\_rc** fields in the verb control block indicate any error.

---

## WinMS()



This is the only entry point supported for Windows 95 and Windows NT.

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);
```

#### Parameter

##### Description

**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 deregisters an application from the management services API.

### Syntax

```
BOOL WINAPI WinMSCleanup(void);
```

### Returns

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

### Remarks

Use **WinMSCleanup()** to indicate deregistration 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()**.

---

## 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);
```

#### Parameter

##### Description

#### 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()**.



---

## 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);
```

### Parameter

#### Description

#### 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 Appendix D of *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.

## WinMSRegisterApplication()

### **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 of Appendix D of *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 than 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.

## **WinMSRegisterApplication()**

**Note:** See also **WinMSUnregisterApplication** and **WinMSGetIndication**.

---

## WinMSUnregisterApplication()

This function deregisters 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);
```

#### Parameter

##### Description

##### 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 Appendix D of *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()**

- **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**.

---

## WinMSGetIndication()

This allows the application to received unsolicited indications.

### Syntax

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

#### Parameter

##### Description

**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**.

---

## 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.

---

## 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.

## TRANSFER\_MS\_DATA

- 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

---

## 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\_RECEIVED**

### **mds\_mu**

MDS\_MU data. The 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.

## SEND\_MDS\_MU

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

---

## 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.

---

## 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

## **NMVT\_RECEIVED**

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.



## Appendix A. IBM APPN MIB Tables

The following table 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.

IBM MIB Table	Node Operator Facility Verb and MIB Table Variables	Input Parameter Settings
<code>ibmappnNodePortTable</code>	QUERY_PORT	<b>port_name</b> <code>ibmappnNodePortName</code>
<code>ibmappnNodePortIpTable</code>	(Note 1)	
<code>ibmappnNodePortDlsTable</code>	QUERY_PORT (select entries with <b>dlc_type</b> of AP_SDLC)	<b>port_name</b> <code>ibmappnNodePortDlsName</code>
<code>ibmappnNodePortTrTable</code>	QUERY_PORT	<b>port_name</b> <code>ibmappnNodePortTrName</code>
<code>ibmappnNodeLsTable</code>	QUERY_LS	<b>ls_name</b> <code>ibmappnNodeLsName</code>
<code>ibmappnNodeLsIpTable</code>	(Note 1)	
<code>ibmappnNodeLsDlsTable</code>	QUERY_LS (select entries with <b>dlc_type</b> of AP_SDLC)	<b>ls_name</b> <code>ibmappnNodeLsDlsName</code>
<code>ibmappnNodeLsTrTable</code>	QUERY_LS	<b>ls_name</b> <code>ibmappnNodeLsTrName</code>
<code>ibmappnNnTopoRouteTable</code>	QUERY_COS	<b>cos_name</b> <code>ibmappnNnTopoRouteCos</code>
<code>ibmappnNnAdjNodeTable</code>	QUERY_ADJACENT_NN	<b>adj_nncp_name</b> <code>ibmappnNnAdjNodeAdjName</code>
<code>ibmappnNnTopologyTable</code>	QUERY_NN_TOPOLOGY_NODE	<b>node_name</b> <code>ibmappnNnNodeName</code> <b>node_type</b> <code>AP_LEARN_NODE</code> <b>frsn</b> 0

IBM MIB Table	Node Operator Facility Verb and MIB Table Variables	Input Parameter Settings
ibmappnNnTgTopologyTable	QUERY_NN_TOPOLOGY_TG	<b>owner</b> ibmappnNnTgOwner <b>owner_type</b> AP_LEARN_NODE <b>dest</b> ibmappnNnTgDest <b>dest_type</b> AP_LEARN_NODE <b>tg_num</b> ibmappnNnTgNum <b>frsn</b> 0
ibmappnNnTopologyFRTable	QUERY_NN_TOPOLOGY_NODE	<b>node_name</b> ibmappnNnFRNode <b>node_type</b> AP_LEARN_NODE <b>frsn</b> ibmappnNnFRfrsn
ibmappnNnTgTopologyFRTable	QUERY_NN_TOPOLOGY_TG	<b>owner</b> ibmappnNnTgFROwner <b>owner_type</b> AP_LEARN_NODE <b>dest</b> ibmappnNnTgFRDest <b>dest_type</b> AP_LEARN_NODE <b>tg_num</b> ibmappnNnTgFRNum <b>frsn</b> ibmappnNnTgFRfrsn
ibmappnLocalTgTable	QUERY_LOCAL_TOPOLOGY	<b>dest</b> ibmappnLocalTGDest <b>dest_type</b> AP_LEARN_NODE <b>tg_num</b> ibmappnLocalTgNum
ibmappnLocalEnTable	QUERY_LOCAL_TOPOLOGY (select entries with unique <b>dest</b> ) (Note 2)	<b>dest</b> ibmappnLocalEnName <b>dest_type</b> AP_END_NODE <b>dest_type</b> AP_LEARN_NODE

IBM MIB Table	Node Operator Facility Verb and MIB Table Variables	Input Parameter Settings
ibmappnLocalEnTgTable	QUERY_LOCAL_TOPOLOGY (Note 3)	<b>dest</b> ibmappnLocalEnTgOrigin <b>dest_type</b> AP_LEARN_NODE <b>tg_num</b> ibmappnLocalEnTgNum
ibmappnDirTable	QUERY_DIRECTORY_LU	<b>lu_name</b> ibmappnDirLuName
ibmappnCosModeTable	QUERY_MODE_TO_COS_MAPPING	<b>mode_name</b> ibmappnCosModeName
ibmappnCosNameTable	QUERY_COS	<b>cos_name</b> 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 information 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  
500 Columbus Avenue  
Thornwood, NY 10594  
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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Department TL3B/062  
P.O. Box 12195  
Research Triangle Park, NC 27709-2195  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copy notice as follows:

(c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. enter the year or years. All rights reserved.

---

## Appendix C. Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM	MVS
Advanced Peer-to-Peer Networking	MVS/ESA
AFP	MVS/XA
AIX	NetView
AnyNet	Operating System/2
APPN	OS/2
AS/400	OS/400
AT	RACF
CICS	System/370
Common User Access	Virtual Machine/Enterprise Systems Architecture
CUA	VM/ESA
IBM	VTAM
IMS	

Other company, product, and service names, which may be denoted by a double asterisk (\*\*), may be trademarks or service marks of others.

C-bus is a trademark of Corollary, Inc.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.



---

# Index

## A

ACTIVATE\_SESSION 181  
activation and deactivation verbs 9  
  ACTIVATE\_SESSION 181  
  DEACTIVATE\_CONV\_GROUP 184  
  DEACTIVATE\_SESSION 186  
  PATH\_SWITCH 189  
  START\_DLC 164  
  START\_INTERNAL\_PU 166  
  START\_LS 168  
  START\_PORT 171  
  STOP\_DLC 173  
  STOP\_INTERNAL\_PU 175  
  STOP\_LS 177  
  STOP\_PORT 179  
ALERT\_INDICATION 626  
alerts, unsolicited 618  
APING 582  
Attach Manager verbs  
  DISABLE\_ATTACH\_MANAGER 596  
  ENABLE\_ATTACH\_MANAGER 597  
  QUERY\_ATTACH\_MANAGER 598

## B

buffer space required 11

## C

CHANGE\_SESSION\_LIMIT 486  
children 28  
common VCB fields 7  
connection network 15, 199  
CPI-C verbs  
  DEFINE\_CPIC\_SIDE\_INFO 587  
  DELETE\_CPIC\_SIDE\_INFO 590  
  QUERY\_CPIC\_SIDE\_INFO 592

## D

data\_lost indicator 13  
DEACTIVATE\_CONV\_GROUP 184  
DEACTIVATE\_SESSION 186  
DEFINE\_ADJACENT\_NODE 28, 124  
DEFINE\_CN 31  
DEFINE\_COS 35  
DEFINE\_CPIC\_SIDE\_INFO 587  
DEFINE\_DEFAULT\_PU 41, 44  
DEFINE\_DLC 46  
DEFINE\_DLUR\_DEFAULTS 50  
DEFINE\_DOWNSTREAM\_LU 52  
DEFINE\_DOWNSTREAM\_LU\_RANGE 55  
DEFINE\_DSPU\_TEMPLATE 58  
DEFINE\_FOCAL\_POINT 61  
DEFINE\_INTERNAL\_PU 65  
DEFINE\_LOCAL\_LU 69  
DEFINE\_LS 74  
DEFINE\_LU\_0\_TO\_3 89  
DEFINE\_LU\_0\_TO\_3\_RANGE 94

DEFINE\_LU\_LU\_PASSWORD 570  
DEFINE\_LU\_POOL 99  
DEFINE\_MODE 101  
DEFINE\_PARTNER\_LU 107  
DEFINE\_PORT 111  
DEFINE\_TP 120  
DEFINE\_USERID\_PASSWORD 572  
DELETE\_CN 126  
DELETE\_COS 128  
DELETE\_CPIC\_SIDE\_INFO 590  
DELETE\_DLC 130  
DELETE\_DOWNSTREAM\_LU 132  
DELETE\_DOWNSTREAM\_LU\_RANGE 134  
DELETE\_DSPU\_TEMPLATE 136  
DELETE\_FOCAL\_POINT 139  
DELETE\_INTERNAL\_PU 141  
DELETE\_LOCAL\_LU 143  
DELETE\_LS 145  
DELETE\_LU\_0\_TO\_3 147  
DELETE\_LU\_0\_TO\_3\_RANGE 149  
DELETE\_LU\_LU\_PASSWORD 574  
DELETE\_LU\_POOL 152  
DELETE\_MODE 154  
DELETE\_PARTNER\_LU 156  
DELETE\_PORT 158  
DELETE\_TP 160  
DELETE\_USERID\_PASSWORD 576  
detailed information 11  
DISABLE\_ATTACH\_MANAGER 596  
DLC\_INDICATION 500  
DLC processes 14  
DLL (dynamic link library) 610  
DLUR\_LU\_INDICATION 501  
DLUS\_INDICATION 504  
DOWNSTREAM\_LU\_INDICATION 506  
DOWNSTREAM\_PU\_INDICATION 511

## E

ENABLE\_ATTACH\_MANAGER 597  
entry points  
  for management services verbs  
    WinMS() 608  
    WinMSCleanup() 609  
    WinMSRegisterApplication() 611  
    WinMSStartup() 610  
    WinMSUnregisterApplication() 614  
  for Node Operator Facility verbs  
    WinAsyncNOF() 19  
    WinAsyncNOFEx() 20  
    WinNOF() 18  
    WinNOFCancelAsyncRequest() 21  
    WinNOFCleanup() 22  
    WinNOFGetIndication() 13, 26, 616  
    WinNOFRegisterIndicationSink() 13, 24  
    WinNOFStartup() 23  
    WinNOFUnregisterIndicationSink() 13, 25

entry points (*continued*)  
  introduction 3, 601

## F

focal point  
  domain 61  
  explicit 61  
  host 61  
  implicit backup 61  
  implicit primary 61  
FOCAL\_POINT\_INDICATION 514  
FP\_NOTIFICATION 627

## G

general protection fault 4, 602

## H

HPR (high-performance routing) 189

## I

indication verbs  
  DLC\_INDICATION 500  
  DLUR\_LU\_INDICATION 501  
  DLUS\_INDICATION 504  
  FOCAL\_POINT\_INDICATION 514  
  LOCAL\_LU\_INDICATION 521  
  LOCAL\_TOPOLOGY\_INDICATION 525  
  LS\_INDICATION 527  
  LU\_0\_TO\_3\_INDICATION 532  
  MODE\_INDICATION 536  
  PLU\_INDICATION 542  
  PORT\_INDICATION 544  
  PU\_INDICATION 546  
  REGISTER\_INDICATION\_SINK\_ 549  
  REGISTRATION\_FAILURE 551  
  RTP\_INDICATION 552  
  SESSION\_FAILURE\_INDICATION 560  
  SESSION\_INDICATION 556  
  UNREGISTER\_INDICATION\_SINK\_ 562  
INITIALIZE\_SESSION\_LIMIT 490  
ISR\_INDICATION 516

## L

limited resource 80  
link stations  
  defined link stations 15  
  dynamic link stations 15  
  implicit link stations 15  
  temporary link stations 15  
list\_options field 11  
  AP\_FIRST\_IN\_LIST 11  
  AP\_LIST\_FROM\_NEXT 11

list\_options field (*continued*)  
  AP\_LIST\_INCLUSIVE 11  
  filtering options 11  
  index value 11  
local descriptor table 4, 602  
LOCAL\_LU\_INDICATION 521  
LOCAL\_TOPOLOGY\_INDICATION 525  
LS\_INDICATION 527  
LU\_0\_TO\_3\_INDICATION 532  
LU pool 90

## M

management services verbs  
  ALERT\_INDICATION 626  
  FP\_NOTIFICATION 627  
  MDS\_MU\_RECEIVED 621  
  NMVT\_RECEIVED 628  
  SEND\_MSD\_MU 623  
  TRANSFER\_MS\_DATA 618  
MDS\_MU\_RECEIVED 621  
MODE\_INDICATION 536

## N

NMVT\_RECEIVED 628  
NN\_TOPOLOGY\_NODE\_INDICATION 538  
NN\_TOPOLOGY\_TG\_INDICATION 540  
node 3  
node configuration verbs  
  DEFINE\_ADJACENT\_NODE 28  
  DEFINE\_CN 31  
  DEFINE\_COS 35  
  DEFINE\_DEFAULT\_PU 44  
  DEFINE\_DEFAULTS 41  
  DEFINE\_DLC 46  
  DEFINE\_DLUR\_DEFAULTS 50  
  DEFINE\_FOCAL\_POINT 61  
  DEFINE\_INTERNAL\_PU 65  
  DEFINE\_LOCAL\_LU 69  
  DEFINE\_LS 74  
  DEFINE\_LU\_0\_TO\_3 89  
  DEFINE\_MODE 101  
  DEFINE\_PARTNER\_LU 107  
  DEFINE\_PORT 111  
  DEFINE\_TP 120  
  DELETE\_ADJACENT\_NODE 124  
  DELETE\_CN 126  
  DELETE\_COS 128  
  DELETE\_DLC 130  
  DELETE\_FOCAL\_POINT 139  
  DELETE\_INTERNAL\_PU 141  
  DELETE\_LOCAL\_LU 143  
  DELETE\_LS 145  
  DELETE\_LU\_0\_TO\_3 147  
  DELETE\_MODE 154  
  DELETE\_PARTNER\_LU 156  
  DELETE\_PORT 158  
  DELETE\_TP 160  
node row (in a class-of-service definition)  
  35

## P

PATH\_SWITCH 189  
PLU\_INDICATION 542

PORT\_INDICATION 544  
ports 14  
  nonswitched ports 14  
  SATF ports 15  
  switched ports 14  
PU\_INDICATION 546

## Q

QUERY\_ADJACENT\_NN 192  
QUERY\_ATTACH\_MANAGER 598  
QUERY\_CN 199  
QUERY\_CN\_PORT 204  
QUERY\_COS 211  
QUERY\_CPIC\_SIDE\_INFO 592  
QUERY\_DEFAULT\_PU 214  
QUERY\_DEFAULTS 216  
QUERY\_DIRECTORY\_LU 225  
QUERY\_DIRECTORY\_STATS 230  
QUERY\_DLC 232  
QUERY\_DLUR\_LU 240  
QUERY\_DLUR\_PU 244  
QUERY\_DLUS 250  
QUERY\_DOWNSTREAM\_LU 254  
QUERY\_DOWNSTREAM\_PU 263  
QUERY\_DSPU\_TEMPLATE 268  
QUERY\_FOCAL\_POINT 272  
QUERY\_ISR\_SESSION 279  
QUERY\_LOCAL\_LU 290  
QUERY\_LOCAL\_TOPOLOGY 298  
QUERY\_LS 303  
QUERY\_LU\_0\_TO\_3 327  
QUERY\_LU\_POOL 337  
QUERY\_MDS\_APPLICATION 341  
QUERY\_MDS\_STATISTICS 344  
QUERY\_MODE 346  
QUERY\_MODE\_DEFINITION 352  
QUERY\_MODE\_TO\_COS\_MAPPING 357  
QUERY\_NMVT\_APPLICATION 360  
QUERY\_NN\_TOPOLOGY\_NODE 363  
QUERY\_NN\_TOPOLOGY\_STATS 369  
QUERY\_NN\_TOPOLOGY\_TG 373  
QUERY\_NODE 380  
QUERY\_PARTNER\_LU 392  
QUERY\_PARTNER\_LU\_DEFINITION 399  
QUERY\_PORT 404  
QUERY\_PU 415  
QUERY\_RTP\_CONNECTION 421  
QUERY\_SESSION 428  
QUERY\_STATISTICS 440  
QUERY\_TP 442  
QUERY\_TP\_DEFINITION 446  
query verbs 10  
  QUERY\_CN 199  
  QUERY\_CN\_PORT 204  
  QUERY\_COS 211  
  QUERY\_DEFAULT\_PU 214  
  QUERY\_DEFAULTS 216  
  QUERY\_DIRECTORY\_LU 225  
  QUERY\_DIRECTORY\_STATS 230  
  QUERY\_DLC 232  
  QUERY\_DLUR\_LU 240  
  QUERY\_DLUR\_PU 244  
  QUERY\_DLUS 250  
  QUERY\_FOCAL\_POINT 272  
  QUERY\_LOCAL\_LU 290

query verbs (*continued*)  
  QUERY\_LOCAL\_TOPOLOGY 298  
  QUERY\_LS 303  
  QUERY\_LU\_0\_TO\_3 327  
  QUERY\_MDS\_APPLICATION 341  
  QUERY\_MDS\_STATISTICS 344  
  QUERY\_MODE 346  
  QUERY\_MODE\_DEFINITION 352  
  QUERY\_MODE\_TO\_COS\_MAPPING 357  
  QUERY\_NMVT\_APPLICATION 360  
  QUERY\_NODE 380  
  QUERY\_PARTNER\_LU 392  
  QUERY\_PARTNER\_LU\_DEFINITION 399  
  QUERY\_PORT 404  
  QUERY\_PU 415  
  QUERY\_RTP\_CONNECTION 421  
  QUERY\_SESSION 428  
  QUERY\_STATISTICS 440  
  QUERY\_TP 442  
  QUERY\_TP\_DEFINITION 446

## R

REGISTRATION\_FAILURE 551  
RESET\_SESSION\_LIMIT 494  
RTP\_INDICATION 552

## S

SATF (shared-access transport facility)  
  15  
security verbs  
  CONV\_SECURITY\_BYPASS 566  
  CREATE\_PASSWORD\_SUBSTITUTE 568  
  DEFINE\_LU\_LU\_PASSWORD 570  
  DEFINE\_USERID\_PASSWORD 572  
  DELETE\_LU\_LU\_PASSWORD 574  
  DELETE\_USERID\_PASSWORD 576  
  SIGN\_OFF 578  
SEND\_MDS\_MU 623  
SESSION\_FAILURE\_INDICATION 560  
SESSION\_INDICATION 556  
session limit verbs  
  CHANGE\_SESSION\_LIMIT 486  
  INITIALIZE\_SESSION\_LIMIT 490  
  RESET\_SESSION\_LIMIT 494  
START\_DLC 164  
START\_INTERNAL\_PU 166, 175  
START\_LS 168  
START\_PORT 171  
STOP\_DLC 173  
STOP\_INTERNAL\_PU 175  
STOP\_LS 177  
STOP\_PORT 179  
summary information 11

## T

TG row (in a class-of-service definition)  
  35  
the Program management services API  
  601  
the Program Node Operator Facility API  
  3

TRANSFER\_MS\_DATA 618

## U

unsolicited alerts 618

## V

verb control block

common fields 7

introduction 3, 4, 601

verbs

activating and deactivating at link level 9

START\_DLC 164

START\_INTERNAL\_PU 166

START\_LS 168

START\_PORT 171

STOP\_DLC 173

STOP\_INTERNAL\_PU 175

STOP\_LS 177

STOP\_PORT 179

activating and deactivating at session level 10

ACTIVATE\_SESSION 181

DEACTIVATE\_CONV\_GROUP 184

DEACTIVATE\_SESSION 186

allowing a management application to "ping" a remote LU 14

APING 582

allowing CPI-C side information to be managed 14

DEFINE\_CPIC\_SIDE\_INFO 587

DELETE\_CPIC\_SIDE\_INFO 590

QUERY\_CPIC\_SIDE\_INFO 592

changing the number of sessions 12

CHANGE\_SESSION\_LIMIT 486

INITIALIZE\_SESSION\_LIMIT 490

RESET\_SESSION\_LIMIT 494

controlling the Attach Manager 14

DISABLE\_ATTACH\_MANAGER 596

ENABLE\_ATTACH\_MANAGER 597

QUERY\_ATTACH\_MANAGER 598

defining resources 8

DEFINE\_ADJACENT\_NODE 28

DEFINE\_CN 31

DEFINE\_COS 35

DEFINE\_DEFAULT\_PU 44

DEFINE\_DEFAULTS 41

DEFINE\_DLC 46

DEFINE\_DLUR\_DEFAULTS 50

DEFINE\_FOCAL\_POINT 61

DEFINE\_INTERNAL\_PU 65

DEFINE\_LOCAL\_LU 69

DEFINE\_LS 74

DEFINE\_LU\_0\_TO\_3 89

DEFINE\_MODE 101

DEFINE\_PARTNER\_LU 107

DEFINE\_PORT 111

DEFINE\_TP 120

deleting resources 9

DELETE\_ADJACENT\_NODE 124

DELETE\_CN 126

verbs (*continued*)

deleting resources (*continued*)

DELETE\_COS 128

DELETE\_DLC 130

DELETE\_FOCAL\_POINT 139

DELETE\_INTERNAL\_PU 141

DELETE\_LOCAL\_LU 143

DELETE\_LS 145

DELETE\_LU\_0\_TO\_3 147

DELETE\_MODE 154

DELETE\_PARTNER\_LU 156

DELETE\_PORT 158

DELETE\_TP 160

description of, how to read 7

common VCB fields 7

returned parameters 7

supplied parameters 7

forcing an RTP connection to switch paths 10

PATH\_SWITCH 189

overview 7

providing security 13

DEFINE\_LU\_LU\_PASSWORD 570

DEFINE\_USERID\_PASSWORD 572

DELETE\_LU\_LU\_PASSWORD 574

DELETE\_USERID\_PASSWORD 576

reporting potential problems to management services focal points 601

ALERT\_INDICATION 626

FP\_NOTIFICATION 627

MDS\_MU\_RECEIVED 621

NMVT\_RECEIVED 628

SEND\_MDS\_MU 623

TRANSFER\_MS\_DATA 618

returning different levels of

information 191

QUERY\_DIRECTORY\_LU 225

QUERY\_DLC 232

QUERY\_DLUR\_LU 240

QUERY\_DLUR\_PU 244

QUERY\_LOCAL\_LU 290

QUERY\_LOCAL\_TOPOLOGY 298

QUERY\_LS 303

QUERY\_LU\_0\_TO\_3 327

QUERY\_MODE 346

QUERY\_MODE\_DEFINITION 352

QUERY\_PARTNER\_LU 392

QUERY\_PARTNER\_LU\_DEFINITION 399

QUERY\_PORT 404

QUERY\_RTP\_CONNECTION 421

QUERY\_SESSION 428

QUERY\_TP\_DEFINITION 446

returning node information in named fields 10

QUERY\_DEFAULT\_PU 214

QUERY\_DIRECTORY\_STATS 230

QUERY\_MDS\_STATISTICS 344

QUERY\_NODE 380

QUERY\_STATISTICS 440

verbs (*continued*)

returning one of more units of

information 10

QUERY\_CN 199

QUERY\_CN\_PORT 204

QUERY\_COS 211

QUERY\_DEFAULTS 216

QUERY\_DLUS 250

QUERY\_FOCAL\_POINT 272

QUERY\_MDS\_APPLICATION 341

QUERY\_MODE\_TO\_COS\_MAPPING 357

QUERY\_NMVT\_APPLICATION 360

QUERY\_PU 415

QUERY\_TP 442

summary 8

unsolicited indications of named

events 12

DLC\_INDICATION 500

DLUR\_LU\_INDICATION 501

DLUS\_INDICATION 504

FOCAL\_POINT\_INDICATION 514

LOCAL-LU\_INDICATION 521

LOCAL\_TOPOLOGY\_INDICATION 525

LS\_INDICATION 527

LU\_0\_TO\_3\_INDICATION 532

MODE\_INDICATION 536

PLU\_INDICATION 542

PORT\_INDICATION 544

PU\_INDICATION 546

registering an application to

receive information 12

REGISTRATION\_FAILURE 551

RTP\_INDICATION 552

SESSION\_FAILURE\_INDICATION 560

SESSION\_INDICATION 556

unregistering an application when it no longer requires information 12

## W

WinAsyncNOF() 19

WinAsyncNOFEx() 20

WinMS() 608

WinMSCleanup() 609

WinMSRegisterApplication() 611

WinMSStartup() 610

WinMSUnregisterApplication() 614

WinNOF() 18

WinNOFCancelAsyncRequest() 21

WinNOFCleanup() 22

WinNOFGetIndication() 13, 26, 616

WinNOFRegisterIndicationSink() 13, 24

WinNOFStartup() 23

WinNOFUnregisterIndicationSink() 13, 25

writing management services programs 602

writing NOF programs 4

# X

XID 78  
XID0 74  
XID3 74



---

## Readers' Comments — We'd Like to Hear from You

eNetwork Communications Server  
Version 6.0 for  
Windows NT and  
eNetwork Personal Communications  
Version 4.2  
for Windows 95 and Windows NT  
System Management Programming

Publication No. SC31-8480-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you?  Yes  No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

Fold and Tape



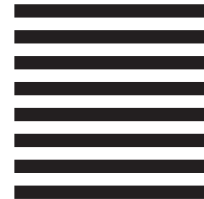
NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

# 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

Cut or Fold  
Along Line





Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SC31-8480-01

